SECURITY AND POLICY REVIEW WORKSHEET- Request for Public Release Clearance

(In Accordance With AFI 35-101, Chapter 15)

(Do not use this worksheet to request clearance of software or web pages.)

#050228

SUBMITTING ORGANIZATION *(Office Symbol)* AFIT/CIGG

A. DOCUMENT TYPE *(Documents must be complete including all figures, charts, photographs and text.)*

☐ ABSTRACT	☐ JOURNAL ARTICLE	☒ ~~THESIS~~ Major Report	☐ DISSERTATION
☐ BROCHURES	☐ NEWS RELEASE	☐ SPEECH	☐ OTHER
☐ CD-ROM	☐ PHOTO WITH CAPTIONS	☐ SUCCESS STORY	
☐ DISPLAY/EXHIBIT	☐ POSTER SESSION	☐ TECHNICAL PAPER	
☐ FACT SHEET	☐ PRESENTATION WITH TEXT	☐ TECHNICAL REPORT	

B. TITLE OF DOCUMENT
An Interactive Robot for Educating College Bound Students About Engineering Disciplines

NO. PAGES: 65

C. AUTHOR(S) NAME AND DUTY TITLE
Jensen, Kent, 2DLT USAF, AFIT Student

OFFICE SYMBOL: AFIT/CIGG

D. FORUM *(Public release clearance is not required for material presented in a closed meeting and which will not be made available to the general public on the Internet or in any print or electronic media)*

DOCUMENT WILL BE ☐ PRESENTED ORALLY	NAME OF CONFERENCE	LOCATION	DATE
DOCUMENT WILL BE ☒ DISTRIBUTED IN PRINT	NAME OF PUBLICATION: University of Utah Library		SUBMITTAL DEADLINE

HAVE RELATED DOCUMENTS BEEN PREVIOUSLY CLEARED FOR PUBLIC RELEASE

☒ NO ☐ YES AFIT CASE NUMBER _____

NOTE: If document will be released to a medium outside the Department of Defense, the following disclaimer must be added: *The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.*

FOR AFIT/PA USE ONLY

Report is approved with the following stipulation: please add "disclaimer" clause.

SUBMIT TO: Air Force Institute of Technology
Public Affairs (AFIT/PA)
Building 642, Room 2010
2950 P Street
Wright-Patterson AFB OH 45433-7765

FOR MORE INFORMATION

Phone: (937) 255-9354 DSN: 785-9354
FAX: (937) 255-2135 DSN: 785-2135
E-MAIL: afit.pa@afit.edu
WEB: http://www.afit.mil/newsrm01.htm

E. NATIONAL SECURITY AND TECHNOLOGY ISSUES

CHECK YES OR NO *(Provide a brief explanation for all positive responses in the SYNOPSIS SECTION below.)*

☐ YES	☒ NO	DOES THE MATERIAL HAVE THE POTENTIAL TO BECOME AN ITEM OF NATIONAL OR INTERNATIONAL INTEREST.
☐ YES	☒ NO	DOES THE MATERIAL AFFECT NATIONAL SECURITY POLICY OR FOREIGN RELATIONS.
☐ YES	☒ NO	DOES THE MATERIAL CONCERN SUBJECTS OF POTENTIAL CONTROVERSY AMONG DOD COMPONENTS OR WITH OTHER FEDERAL AGENCIES.
☐ YES	☒ NO	DOES THE MATERIAL CONTAIN TECHNICAL DATA DEVELOPED UNDER CONTRACT OR INDEPENDENTLY DEVELOPED AND CONTROLLED BY THE INTERNATIONAL TRAFFIC IN ARMS REGULATIONS (ITAR) THAT MAY BE MILITARILY CRITICAL AND SUBJECT TO LIMITED DISTRIBUTION, BUT ON WHICH A DISTRIBUTION DETERMINATION HAS NOT BEEN MADE.
☐ YES	☒ NO	DOES THE MATERIAL CONTAIN INFORMATION ON NEW WEAPONS OR WEAPON SYSTEMS, SIGNIFICANT MODIFICATIONS OR IMPROVEMENTS TO EXISTING WEAPONS OR WEAPON SYSTEMS, EQUIPMENT OR TECHNIQUES.
☐ YES	☒ NO	DOES THE MATERIAL CONTAIN INFORMATION ON NATIONAL COMMAND AUTHORITIES; COMMAND, CONTROL, COMMUNICATIONS, COMPUTERS AND INTELLIGENCE; INFORMATION WARFARE; OR COMPUTER SECURITY.
☐ YES	☒ NO	DOES THE MATERIAL CONTAIN INFORMATION ON MILITARY ACTIVITIES OR APPLICATIONS IN SPACE, NUCLEAR WEAPONS, INCLUDING WEAPON-EFFECTS RESEARCH; CHEMICAL AND BIOLOGICAL WARFARE ISSUES; BIOLOGICAL AND TOXIN RESEARCH, HIGH-ENERGY LASERS AND PARTICLE BEAM TECHNOLOGY; ARMS CONTROL TREATY

F. SNYNOPSIS *(Provide a brief description of the system, process, or technology, including its state of development and whether the application is military, commercial, or dual-use).* EXAMPLE: This is new concept of applying current development for high-powered lasers, specifically for military applications.

G. REFERENCES *(Check all that apply)*

☒ ALL REFERENCES ARE UNCLASSIFIED, UNLIMITED, AND ARE AVAILABLE TO THE PUBLIC

☐ REFERENCE # ARE SUBJECT TO DISTRIBUTION LIMITATION. NO LIMITED INFORMATION FROM THESE REFERENCES IS INCLUDED IN THE DOCUMENT

☐ NO REFERENCES ARE INCLUDED IN THIS DOCUMENT

RELEASE AUTHORITY AND MANAGEMENT RESPONSIBILITY FOR JOINT EFFORTS

IS THIS INFORMATION CO-AUTHORED WITH SOMEONE FROM AN ORGANIZATION OUTSIDE OF AFIT.

☐ YES ☒ NO

DOES YOUR ORGANIZATION HAVE THE AUTHORITY TO RELEASE THE INFORMATION

☒ YES ☐ NO

IF WORK IS SPONSORED BY ANOTHER ORGANIZATION, DO YOU HAVE THE AUTHORITY TO RELEASE ALL THE INFORMATION

☒ YES ☐ NO

IF NOT, WHAT ORGANIZATION SHOULD COORDINATE ON THE REQUEST

ORGANIZATION	PHONE

NOTE: If the organization is a DOD organization, AFIT/PA will coordinate release approval with that organization's PA office

I.	AUTHOR/TECHNICAL REVIEW CERTIFICATION

AUTHOR

I certify the information contained in the attached document is technically accurate and does not disclose classified, sensitive, or militarily critical technology and does not violate proprietary rights or copyright restrictions. All security and technology issues listed above have been considered and any applicable security classification guides have been reviewed.

SIGNATURE: NOT AVAILABLE	DATE
AUTHOR(S) NAME (Print): Kent Jensen	TELEPHONE
OFFICE SYMBOL/ORGANIZATION: AFIT/CIGG	E-MAIL ADDRESS: jensenkr@hotmail.com

TECHNICAL REVIEWER CERTIFICATION (Faculty Advisor)

I certify the information contained in the attached document is technically accurate and does not disclose classified, sensitive, or militarily critical technology and does not violate proprietary rights or copyright restrictions. All security and technology issues listed above have been considered and any applicable security classification guides have been reviewed.

SIGNATURE	DATE
TECHNICAL REVIEWER'S NAME (Print)	TELEPHONE
OFFICE SYMBOL/ORGANIZATION	E-MAIL ADDRESS

DEPARTMENT HEAD *(or equivalent)*/PROGRAM MANAGER CERTIFICATION

The attached material submitted for public release has been properly staffed and reviewed by this division/organization. I support the recommendation of the Author and Technical Reviewer for public release.

SIGNATURE: Melissa L. Flatley	DATE: 26 Feb 02
NAME *(Print)*: MELISSA L FLATTERY	TELEPHONE: 5-3152 x3026
OFFICE SYMBOL/ORGANIZATION: AFIT/CIGG	

AFIT PUBLIC AFFAIRS

THE ATTACHED MATERIAL IS/IS NOT CLEARED FOR PUBLIC RELEASE

SIGNATURE OF SECURITY AND POLICY REVIEW OFFICER: Pat Wagner	DATE: 1 Mar 02

J. REMARKS *(please reference the specific section)*

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

1 March 2002

MEMORANDUM FOR AFIT/CIGG (Capt Flattery)

FROM: AFIT/PA Security and Policy Review

SUBJECT: Your Requests for Public Release Approval,
"An Interactive Robot for Educating College-Bound Students about Engineering Disciplines"
By: 2nd Lt Kent Jensen

CASE Number: AFIT/PA Case Number 020228

1. The report titled above has been reviewed and is cleared by AFIT/PA for public release with the following stipulation: please add "disclaimer" clause.

2. If you have any questions, please call me at DSN: 785-2216.

PAT WAGNER,
Public Affairs Specialist

An Interactive Robot for Educating College Bound Students About Engineering Disciplines

Kent Jensen

3 Aug 2001

University of Utah Mechanical Engineering Department

Salt Lake City, UT

THE VIEWS EXPRESSED IN THIS ARTICLE ARE THOSE OF THE AUTHOR AND DO NOT REFLECT THE OFFICIAL POLICY OR POSITION OF THE UNITED STATES, DEPARTMENT OF DEFENSE, OR THE U.S. GOVERNMENT

Abstract

The objective of this project is to design and build a ball-throwing robotic arm for the purpose of high school visits. When taken to local high school physics classes, this ball-throwing arm experience will present college bound high school students with a series of activities that will allow them to interact with typical challenges found in Physics, Mechanical Engineering, Electrical Engineering, and Computer Science. Stimulating interest and understanding of engineering challenges amongst younger students is critical in helping them make informed decisions affecting future careers in engineering. The key challenges in this project include: 1) designing an appropriate robotic arm; 2) building it; 3) creating a computer implemented control system which can converge two positions and two velocities at a pre-specified instant in time; and 4) preparing exciting interactive presentations and support materials that will effectively educate high school students about their career interests.

Acknowledgments

The University of Utah proved to be a wonderful place for me to work on my Master's of Engineering. They cooperatively worked with me to fit their program to the constraints the Air Force placed on me. I received a great deal of help from many different people, who gave of their time to help me succeed in one way or another. I wanted to especially thank those listed below.

- Dr. Robert Roemer provided the impulse and funding for this project. He had the vision of creating an interactive robot to use with high school students. Without him, this would have never started.
- Dr. Mark Minor served as my graduate advisor, contributing multiple hours weekly advising this project, helping with everything from homework advice to the design, construction and implementation of the Ball-Throwing Robot.
- Dr. Sanford Meek had me in more classes than any other instructor. He taught nearly half of all my credits. He practically taught me all of what I know about control systems.
- Tom Slowik from the Professional Machine Shop gave hours and hours of help to me while I machined all but a few of the parts in his shop.
- Danny Blanchard worked diligently for a semester on the robot design and worked to program the PK2600's controller processor.
- Roy Merrell has more experience than any of the rest of us using the PK2600. He broke the code, figuring out how to create a user interface for the robot

that would allow the PK2600 display processor to work with the controller processor.

- Julia Anderson helped machine parts and build the quadrature decoding circuits.
- ME Office Secretaries, Dana, Donna, Debi & Sylvia, who helped me figure out how to get the paper work done, such that my project could go forward.

Table of Contents

Table of Tables

Table of Figures

1 Introduction

1.1 Motivation

Stimulating interest and understanding of engineering challenges amongst high school students is critical in helping them make informed decisions about pursuing careers in engineering. The objective of this project is to present the students with a series of activities that would introduce them to typical challenges found in Mechanical Engineering, Electrical Engineering, and Computer Science. These activities are based on a robotic arm designed to throw a ball at a specific target. The students will learn about measurement techniques, particle kinematics, robotics and teamwork.

The students will first learn about a ball traveling through the air via tutorials on particle kinematics that will be presented in their physics class. This knowledge will then be used to determine what initial velocities are required to throw the ball at a series of targets. A representative from the University of Utah will then visit the class and bring a self-contained ball-throwing robot. The representative will illustrate the tasks faced by the engineering disciplines while explaining the robot's design, electrical and mechanical systems, micro-controller, and performance limitations. The student teams will then setup targets throughout their classroom, measure the distances to the targets from the robot, and calculate appropriate ball speeds and trajectories considering the robot limitations and room dimensions. Each team will then program their results into the robot and see if they can hit the targets. By awarding points for accuracy and creativity, such as an off-the-wall shot, a fun and competitive environment would be encouraged.

The main focus of this project is to present a fun and exciting example of engineering. Through a basic physics lesson, the students will use their engineering skills

to solve a real problem and have the unique hands-on opportunity to test their results on a robotic system. This should provide the potential engineers with a glimpse of the exciting challenges that engineers face and help them make educated career choices.

1.2 Objectives for my Master's of Engineering Degree

My project is to build a robot that can fill such a role. The objectives for the interactive presentation set the minimum capabilities of the robot. The robot must be capable of throwing a ball to any commanded position in the room. This implies motors with a specific performance range, a computer controller capable of converging two positions and two velocities simultaneously, and hardware designed to be accurate, long lasting and easily transportable. These various objectives each influenced the design of the robot in a different way.

Obviously setting such expectations early on in the course of this degree set the tone of the coursework taken. Each class had the purpose of teaching skills essential to the completion of this project. The courses taken with this purpose in mind included:

Table 1 Class Schedule

Class Description	Class Number	Credit Hours
ME EN Advanced Control Systems	6200	3.0
ME EN Robotics	6220	3.0
ME EN Optimal Controls	7210	3.0
PHYCS Electronics I	5610	4.0
PHYCS Electronics I Lab	5610	0.0
ME EN Compensator Design	6210	3.0
ME EN Intermediate Dynamics	6410	3.0
ME EN Nonlinear Controls	7200	3.0
PHYCS Electronics II	6620	4.0
PHYCS Electronics II Lab	6620	0.0
ME EN Special Project	6970	4.0

1.3 Structure of Paper

This project report is written to a level that will be instructive to high school students. Obviously many of the modeling and controls details are beyond the scope of high school physics and this paper cannot provide sufficient background to fully explain these tools. However, this report is intended to fulfill two roles, one as a source of information for students and secondly as a project report and technical background to the robot.

The report begins by explaining the objectives and some of the background in the subject. The remaining portion tells the story of the robot's development, starting with the design, explaining many of the reasons certain design characteristics were chosen. Once construction and assembly has taken place it then outlines the controller prototyping and testing of the system.

2 Ball-Throwing Activity

The whole purpose of this entire project is to interact with potential students in an exciting way, inform them about engineering careers and guide those with interest into engineering programs at the University of Utah. In order for the students to have the best experience they need to have enough background to be proficient in the skills needed for the activity and the activity needs to challenge them in a way not usually done. This section outlines the envisioned Ball-Throwing Robot experience from start to finish as illustrated in Figure 1.

When a potential class has been identified, a University of Utah representative will contact the teacher and discuss the skills needed by the students to participate. An information packet will have been prepared, which outlines the necessary skills from start

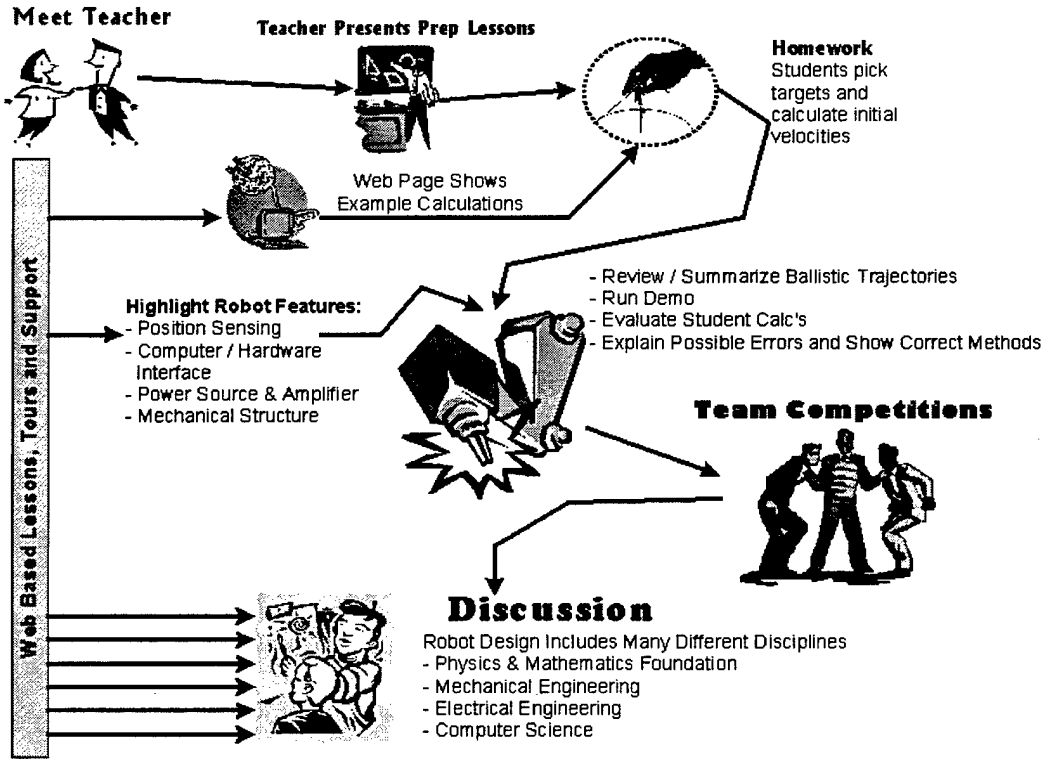


Figure 1 Flow Chart Describing the Course of the Demonstration

to finish and has pre-worked examples, which give examples of all basic situations encountered. The teacher will either teach or review the material to make certain that the students understand the subject matter. For the students' homework the teacher would assign the task of setting out target locations around the room and calculating the appropriate robot commands – base angle and release velocity – needed to hit their target. A web-site will also contain instructive material, showing how to get the best success with the robot, discussing how release angle affects motor saturation or how arrival angle influences the probability of a direct hit. The web site could also handle extra credit problems like how to bounce off the floor or a wall and hit a target.

The next stage of the experience will be the actual visit, when the U of U representatives bring the robot to the school. The representative will point out various features of the robot and briefly explain how it works, highlighting design issues, which may be discussed further, in the end of the demonstration. In order to validate the robot's ability, the U of U representative will demonstrate a successful ball-throwing sequence or series of sequences. At this point the student solutions will be evaluated for accuracy. Then comes the competition.

The students will divide into teams and compete against each other for accuracy and difficulty. They will need to measure the distances to the targets and calculate the release velocities and angles. Extra points will be given to the team that gets the answer fastest only if their throw is successful. If two robots are available for a single demonstration, then the students can compete in time for the same targets. Furthermore extra points will be given for teams that incorporate a bounce into their throw or use a lightweight ball and successfully account for aerodynamic drag.

The final phase of the demonstration follows the competition and involves a more in depth discussion of what makes the robot work. The discussion will explain how different engineering disciplines work together to create something like this robot. The representative could explain that:

- a mechanical engineer would be involved in design of the robot, modeling of its dynamics and control of its movements;
- a computer scientist would be involved in creating the user interface, the simulation programs, and the controlling programs which implement the algorithms the mechanical engineer designs; and
- an electrical engineer is concerned with interfacing the computer controller with the actual robot without frying any chips, making sure the power source, amplifiers, encoders and computer are all communicating properly and making sure no one gets hurt by the electrical components of the robot.

This discussion should be one focused on getting the students to talk, ask questions and learn what each different field contributes to engineering projects, such that the components can work together as a whole.

The classroom discussion will be limited on time and won't be able to explain all of the various aspects in depth. Material explaining how the robot works will also be placed on the web site for students to access and learn more about the various aspects of its design, construction and operation.

3 Background

A survey of similar projects uncovers many projects involving common threads, but different objectives. There are many robots in the literature that throw objects, using various throwing movements. They span a broad range of type and purpose. The simplest start as single degree of freedom throwing robots built both commercially and for research purposes. The next common category is the two-link category, which includes traditional rigid planar pairs and flexible member manipulators. Many robots have more degrees of freedom and have been built to catch as well as throw balls and even juggle multiple balls. The juggling robots encompass their own group, varying broadly in approach and form. The ideas encompassed in these differing projects do encounter similar challenges in implementation.

Michael J. Northrop of Northwestern University [1] uses a one-link robot to feed parts in an assembly line by throwing them and manipulating their landing position and orientation. Once the part is identified and the trajectory planned, then the motor must execute the command precisely.

Eric W. Aboaf [2] of MIT explored the advantage of task-level control when throwing balls with a one-link robot. The system uses a vision system to measure the ball's actual landing position, which can then be compared against the desired landing position. They explored two methods of using this error to improve performance. Their Fixed-Model method learns by applying an inverse model to the result. Their Refined Model procedure manipulates both the model and the command signal to get the desired results. It would be interesting to build a self-monitoring and correcting system into this robot.

Norihiko Kato [3] of Mie University in Japan explores the control of a two-link planar pair robot. He implements an adaptive non-linear controller that modifies the release time. He shows improved results through release time manipulation.

The Ball-Throwing Robot shares objectives and requirements with each of these projects. One feature not shared is that the Ball-Throwing Robot has a goal of converging two positions and two velocities at one fixed point in space and time. The others are flexible in their release and catching points. The control and implementation strategies of each project mentioned are adapted specifically to its purpose. Similarly each of these provides hints to accomplish these specific objectives, but none are identical.

4 Robot Mechanical Structure

4.1 General Format

The Ball-Throwing Robotic Arm is a three-link manipulator, depicted in Figure 2. It has two axes of shoulder rotation, an elbow and a simple open-close two-fingered end effector or hand. This configuration was chosen because of the constraints implied by the objectives of the demonstration: a fixed release point; a large range of release velocities; and a quick releasing hand mechanism.

The purpose of this robot is to interact with high school students. If the objective

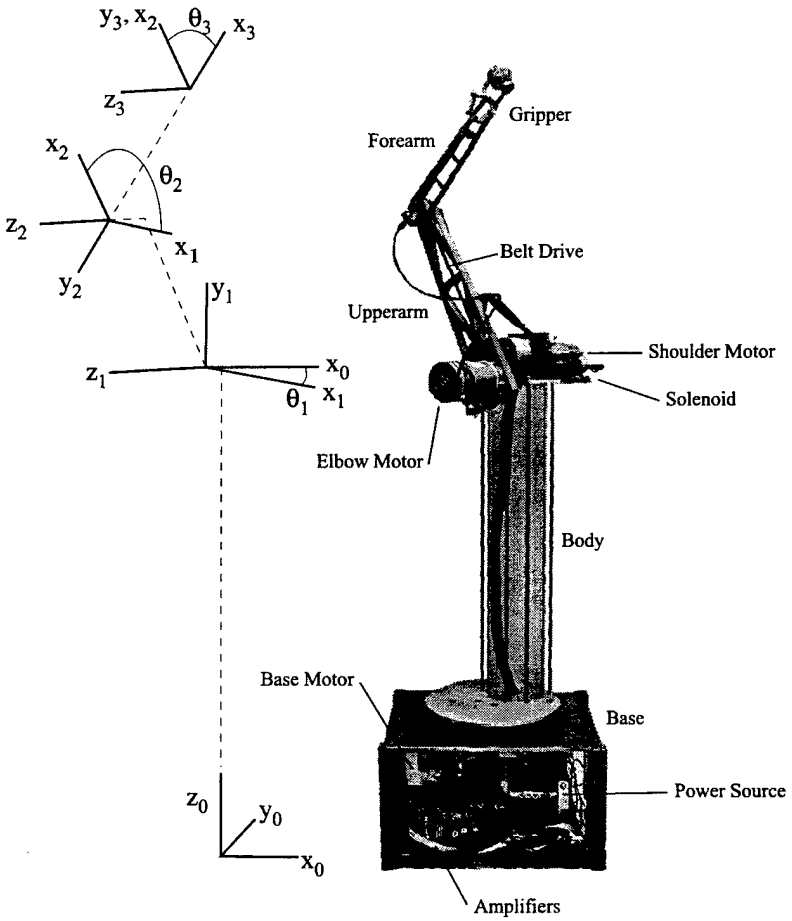


Figure 2 ProEngineer Generated Image of the Assembled Ball-Throwing Robot

were just to hit an arbitrary target in the room, then a two-degree of freedom robot with an end effector would be sufficient as is illustrated in Figure 3a. The simplest ball-throwing robot would have one link and release from one point with variable speed. Although feasible and practical, this would not be exciting for the students and would constrain the problem so much that the mathematical calculations would be nearly trivial.

A slightly more complex mode would be a one-link robot that could release from any point along an arc to achieve the desired range (Figure 3b). Such a configuration would allow the students freedom in choosing angle and release velocity, but in addition, each release angle variation would also change the release point. Although straightforward to solve with a system of equations, some high school students may not have been exposed to such methods and may find it difficult to solve, without arbitrarily choosing one of the constraints.

A third method involves a three degree of freedom, two-link robot. A two-link robot has the ability to theoretically achieve any velocity at a single point within its

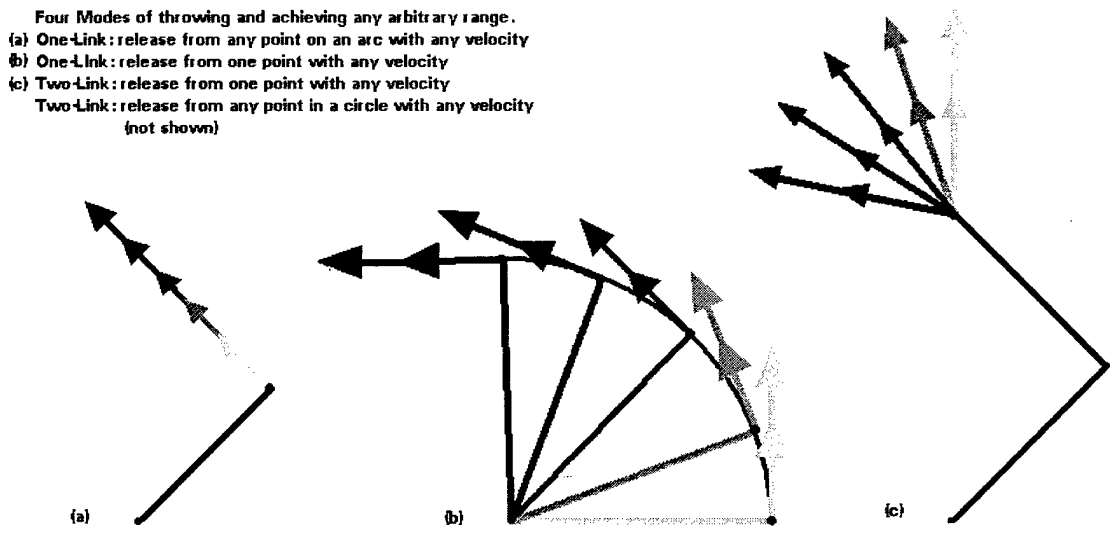


Figure 3 Various Ball-Throwing Methods That Could Have Been Used

workspace through a linear combination of the two joint velocities (Figure 3c). This allows for the choice of the release point by the programmer (choosing a point with a large range of possible throws), somewhat simplifying the students' problem, but still allowing them full choice of velocity vectors. Such freedom allows for discussion of the advantages of different angles with respect to flight time, air drag and robot actuator saturation.

Allowing students to choose release point and release velocity introduces another level of difficulty. Such a problem would have to deal with actuator saturation in addition to previous considerations. Although relevant and applicable, the far too open-ended calculations would be beyond the scope of a high school physics class. For these reasons a three degree of freedom, two-link robot was chosen, which, although capable of any release point, releases the ball from the same point every time it throws.

4.2 Actuator Selection

The decision to build a ball-throwing robot established some basic criterion from which the design process began. The main objective was to create a robot capable of throwing balls to designated targets within a classroom size area. This was more specifically quantified as a maximum distance of 6-8 meters. This requires specific joint velocity combinations in order to achieve such distances. Therefore the first step taken in the design process was to calculate some example joint velocity combinations and the correlating distances that the ball would travel when released with those initial conditions.

Some of the example calculations are shown in _. These calculations gave us a baseline from which to begin torque calculations, which would give us an estimate of the motor sizes such a robot would require in order to meet the stated objectives.

The next logical step in the design was to estimate some basic dimensions and masses of the robot arm links, such that a dynamic model of the robot could be

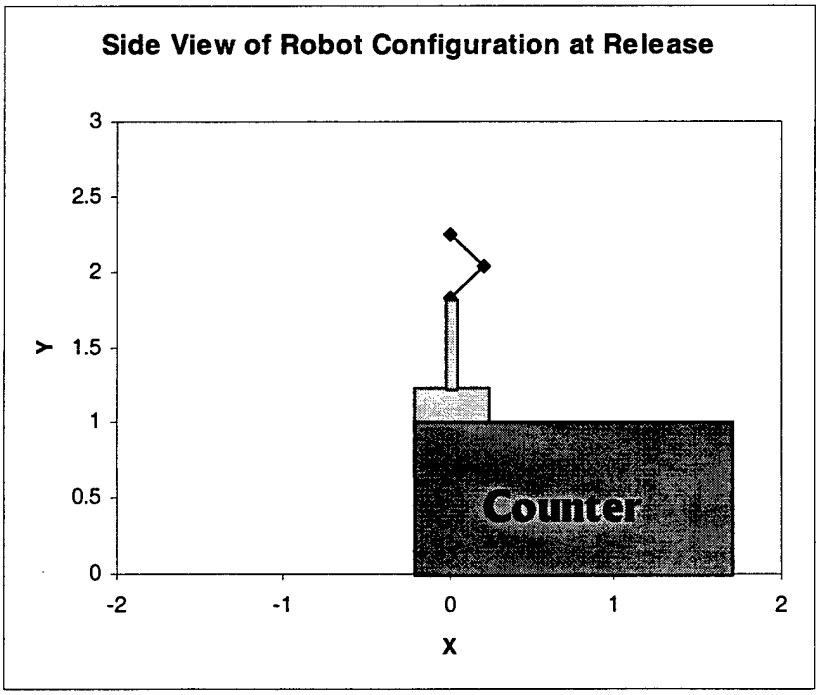
Table 2 Sample Values Used for Torque Estimates and Motor Sizing

Upper Arm Length	Theta 2 @ Release	Fore Arm Length	Theta 3 @ Release
0.3	2.35	0.295	1.57

	X Location	Y Location
Base	0	1.82
Elbow Location	0.21	2.04
End Effector Location	0.003	2.25

Release Vx	Theta 2 Dot	Release Vy	Theta 3 Dot
2	2.36	3	-14.34
2	4.71	4	-19.10
2	7.07	5	-23.85
3	-7.07	0	-0.12
3	-4.71	1	-4.87
3	-2.36	2	-9.63
3	0	3	-14.38
3	2.36	4	-19.14
3	4.71	5	-23.89

Dep. Angle	Time Aloft	Distance
56.31	1.05	2.10
63.43	1.20	2.40
68.20	1.36	2.72
0	0.68	2.03
18.43	0.79	2.36
33.70	0.91	2.74
45	1.05	3.15
53.13	1.20	3.60
59.04	1.36	4.08



mathematically formulated, from which required torques could then be calculated. Initially a Lagrange-Euler Formulation of a two-link manipulator was used [4]. The model assumed: equal length forearm and upper arm; no friction and centers of gravity located at the midpoint of each link. The resulting equations of motion are shown in Equation (1).

From this point only angular position, velocity and acceleration trajectories would be needed to calculate the estimated motor torques necessary for a two-link manipulator to effect such motions.

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}=\begin{bmatrix}\frac{m_1l^2}{3}+\frac{4m_2l^2}{3}+m_2C_2l^2 & \frac{m_2l^2}{3}+\frac{m_2l^2C_2}{2}\\ \frac{m_2l^2}{3}+\frac{m_2l^2C_2}{2} & \frac{m_2l^2}{3}\end{bmatrix}\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}$$
$$+\begin{bmatrix}-\frac{m_2S_2l^2\dot{\theta}_2^2}{2}-m_2S_2l^2\dot{\theta}_1\dot{\theta}_2\\ \frac{m_2S_2l^2\dot{\theta}_1^2}{2}\end{bmatrix}+\begin{bmatrix}\frac{m_1glC_1}{2}+\frac{m_2glC_{12}}{2}+m_2glC_1\\ \frac{m_2glC_{12}}{2}\end{bmatrix} \qquad (1)$$

C1 designates $\cos(\theta_1)$, S12 designates $\sin(\theta_1+\theta_2)$, etc.

Two different methods of trajectory determination were considered during these initial stages. One method attempted to minimize individual motor torques by calculating the smoothest acceleration and deceleration combinations for the joints individually. The other method used the Jacobian and inverse kinematics to calculate the path of the end effector, which can result in larger accelerations of the individual joints. The objective of these preliminary calculations was to establish a baseline torque requirement, which would designate the minimum motor sizes required to meet the objectives. Since the objective was to find the approximate minimum torques needed, the first trajectory calculation method was used.

Each joint had an initial position, initial velocity, release position, release velocity, stop position, and stop velocity specified. This information allowed for easy calculation of trajectories with smooth accelerations of the individual joints via cubic spline interpolations. A cubic spline was calculated between the initial and release values and a second between the release and stop values. The calculation of the cubic spline provided the required angular position, velocity and acceleration values needed to calculate motor torques τ_1 and τ_2. A Matlab m-file containing these equations produced the following torque curves for joints one and two as estimates of the required torques. Notice in Figure 4 that the example release angular velocities, 4 r/s and –21 r/s correlate to x and y release velocity values of 3 m/s and 5 m/s respectively. This correlates to a

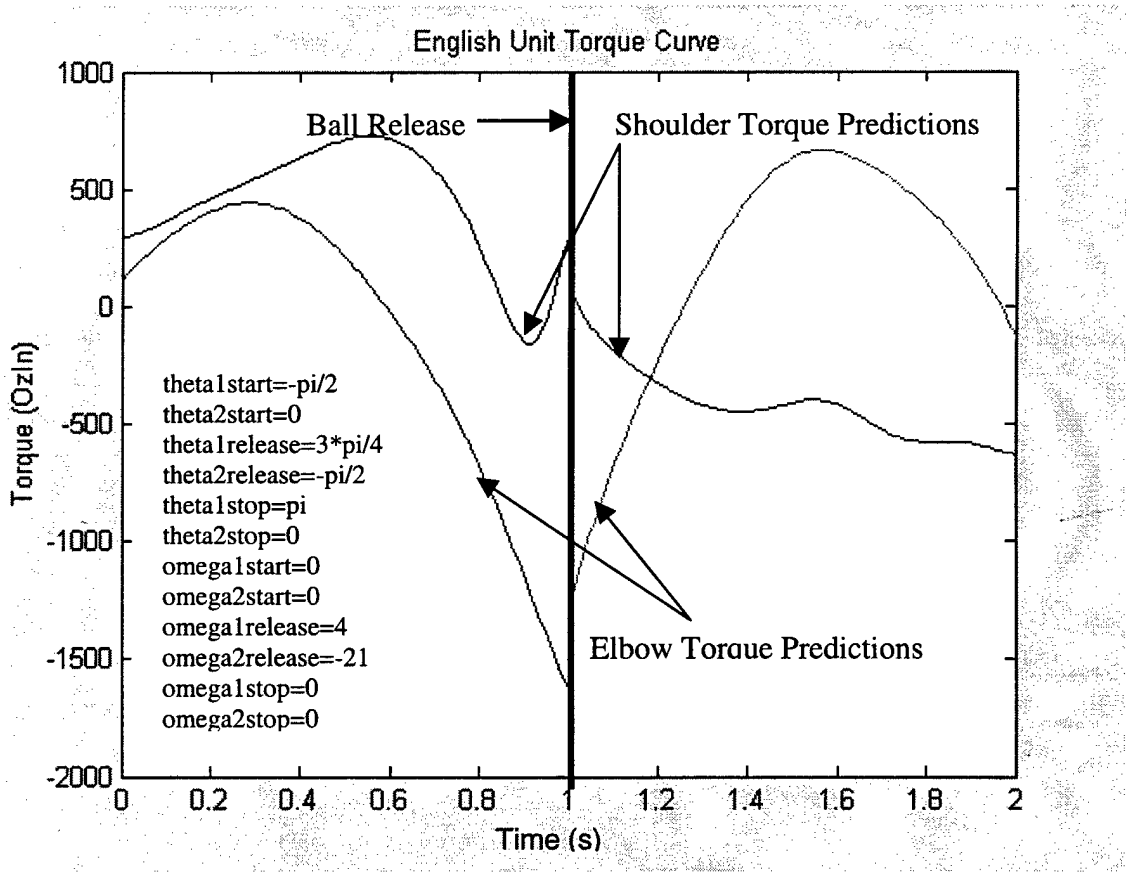


Figure 4. Torque Curves for a Simulated Throwing Motion

range of ~8 meters from _. Similar such calculations and data sets provided a basis of evaluation for motors in the marketplace.

At this point enough information was known to begin shopping for motors. In order to accomplish the objectives the robot would need to be capable of: moving at least 20 r/s (191 rpm); exerting 1500 OzIn of torque at 20 r/s; and be affordable. The 1500 OzIn at 20 r/s occurs as a spike in the torque trajectory, therefore the motor would not have to be capable of operating continuously at such high levels. After surveying the possibilities, surplus Matsushita motors from Servo Systems were found to be the best combination of characteristics needed for this robot. Figure 5 shows the specifications of the Matsushita motors purchased from Servo Systems. The 24 V data is that which was taken in the Servo Systems laboratory. They tested at 24 V, and therefore were not sure

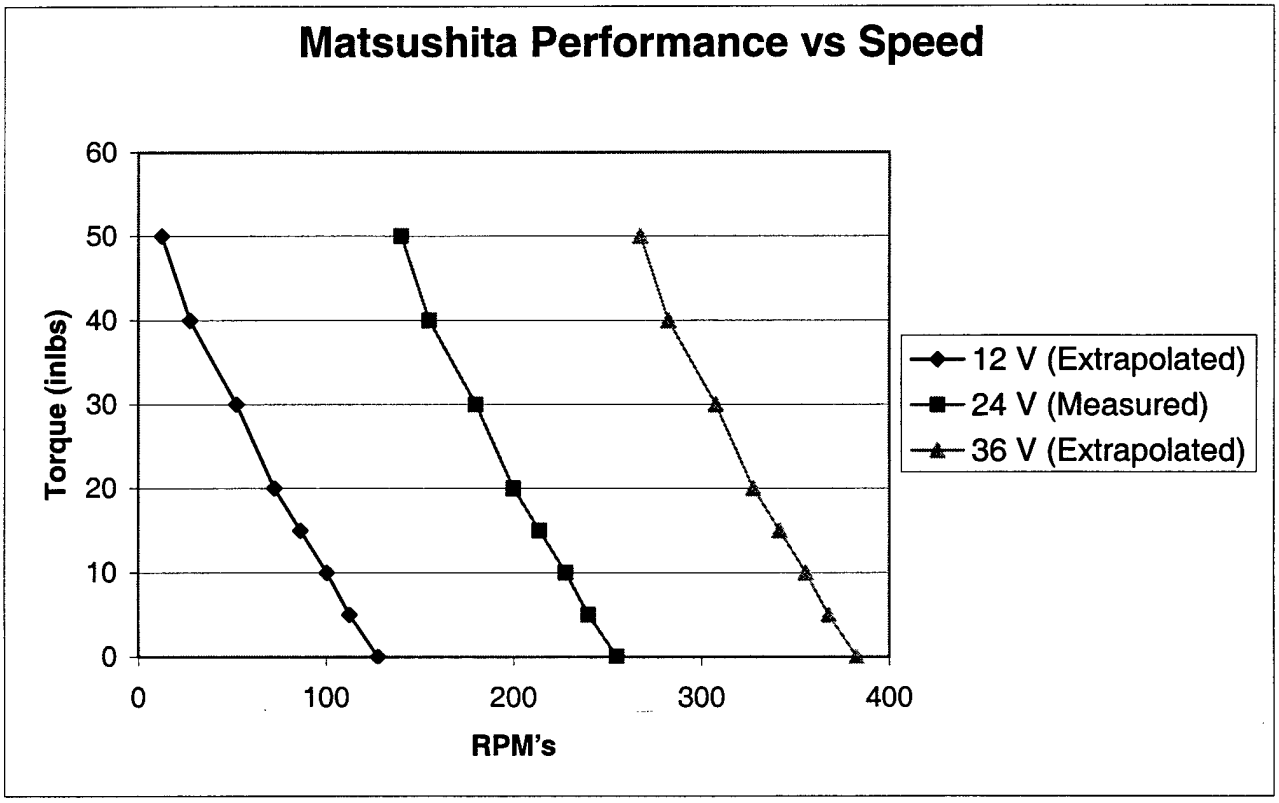


Figure 5 Matsushita Motor Performance Data Published by Servo Systems Co

of the maximum motor ratings. This chart shows extrapolated torque vs. speed curves for 12 V and 36 V operations based on measurements of the 24 V operations. Although at 24 V the motor does not exert enough torque, the extrapolated torque vs. speed curve of the 36 V performance shows a predicted torque exceeding the levels needed for the ball-throwing robot arm.

The extrapolations are based on the idea that a motor has a characteristic resistor value, back-emf constant and torque constant. Equations (2) through (4) express the electrical relationship of the voltage source, resistor and motor, in a steady state mode.

$$V_s = IR + k_e\omega \tag{2}$$

$$I = \frac{k_e\omega - V_s}{R} \tag{3}$$

$$T = k_t I = \frac{k_t k_e}{R}\omega - \frac{k_t}{R}V_s \tag{4}$$

In such a mode of operation, the torque constant, back-emf constant and resistor values establish the slope of the torque speed curve and the supply voltage sets the intercept

Servo Systems recommended 24V usage, but did not have the original motor specifications, therefore did not know the current limitations of the motors. Their 24V tests never destroyed the motors, therefore they published that as a safe operating level. They did not publish an operating level that did destroy the motors, therefore 36 V operation is unknown.

Running the motor at 36 V does introduce the possibility of running too much current through the windings, melting the wires' insulation and destroying the motors in the process. The torque trajectory simulations shows that a commanded torque at levels requiring 32 V operation occurs for only a fraction of a second. Such brief bursts of power exceeding recommended operating modes were considered acceptable because of

the brief application period. Experience shows that the higher voltage works fine, but it will probably reduce the life of the motor.

Four such Matsushita motors were purchased from Servo Systems for use in the Ball-Throwing Robot Arm Project. Reasons for purchase included: the lower price of the surplus motors compared to other motors with similar performance capabilities; the inclusion of optical encoders with a resolution of 1850 counts/rev (due to the 1:18.5 ratio gearhead); and the nearly ideal performance fit.

Once the motors were chosen and their potential power consumption levels were estimated the choice of power amplifiers and power sources was straightforward. Servo Systems again provided the associated current levels with each data point of the 24 V operation regime. Extrapolation from this data predicted that the motor would require approximately 11 A of current at 32 V in order to provide the necessary torque. Advance Motion Control builds a servo amplifier, which can supply a peak current of 12 A (for 2 seconds max) and 6 A continuous current, using a power source of 20V-80V. This servo amplifier closely fit our needs, so we purchased three of these and a power source, which can supply 40V at 40A continuously. The power supply can supply all three motors at maximum required performance. Such power consumption is not expected, but the equipment is capable of handling it.

4.3 Design Considerations

The following section is included to explain the reasons for which the design choices were made, which resulted in the current Ball-Throwing Arm design. It outlines how the design was developed as well as why various decisions were made. The Engineering Sketches of all parts designed on ProEngineer are included in Appendix A.

The software ProEngineer was used to design and model the robot in a virtual environment before it was actually built. The objective in this effort was to fully design and assemble the robot in a virtual environment before ever making a piece. Such virtual assembly would hopefully expose myopic designs, integration conflicts, and avoid time intensive redesigns later on. This also allowed for the sizing and counting of bolts and connectors beforehand, such that the appropriate bolts and connectors could be purchased ahead of time. Viewing the parts before construction also provided the opportunity to determine whether or not the parts were realistically machinable. Although the Haas CNC machine was available for use, making pieces by hand proved much more expeditious than making tooling plates and G-Code for all of the parts.

The Ball-Throwing Robotic Arm must be easy to transport and move, therefore all of the power supplies, power amplifiers and controlling computer should be compact and centrally located. For this reason the heavy power supply and power amplifiers were

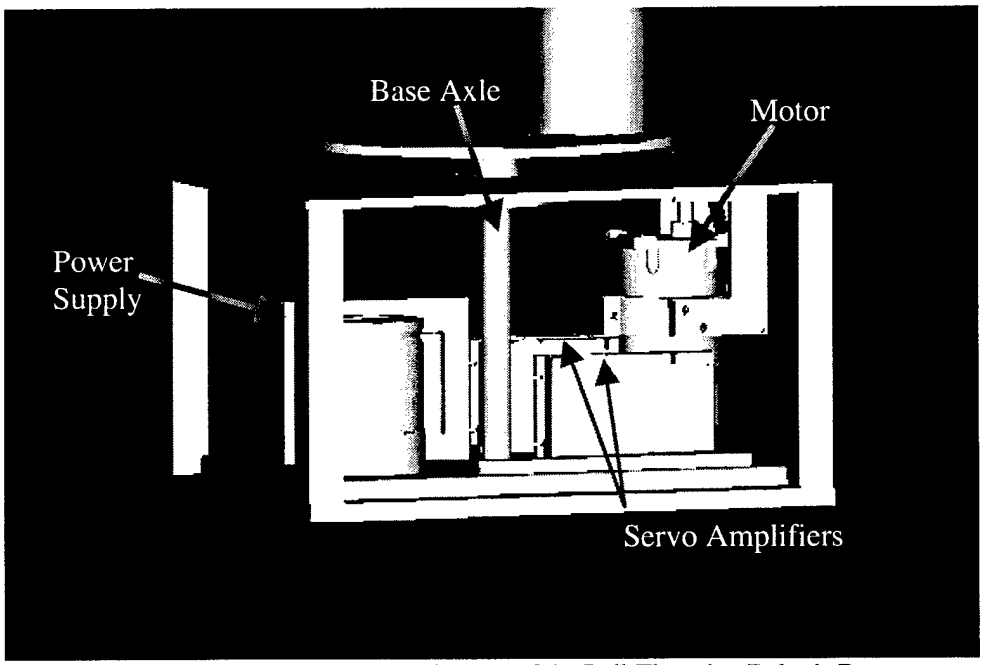


Figure 6 ProEngineer generated image of the Ball-Throwing Robot's Base

used as ballast to hold the arm steady during throwing motions as shown in Figure 6. A hole was drilled through the center of the power supply base plate, through which the main axle was mounted. The axle was mounted using two sets of tapered-roller bearings, similar in fashion to the mounting of a car wheel. This configuration constrained five of six potential degrees of freedom, preventing it from shifting out of line during transportation or throwing motions. The base plate upon which all other parts were mounted was made of ½" aluminum versus the top plate, which was ¼" aluminum. The top plate was designed to flex, in order to place pressure on the tapered-roller bearings, preloading the bearings, increasing structural stiffness. Since the base plate was twice as thick as the top plate, it has significantly less flex than the top plate, therefore maintaining a flatter base. Furthermore the steel base of the power supply is bolted to the ½" aluminum plate in four places, providing further bending strength.

The motor used to rotate the base can also be seen in Figure 6. This motor was placed between servo amplifiers with the intent of compacting the design as much as possible. This motor is connected to the axle via a timing belt and timing pulleys, to prevent drift and minimize backlash. Due to the nature of any belt, the design must include a mechanism for tensioning the drive belts. The base motor is encased in a circular mount, which tightens around the motor casing to fix it in place. This particular motor uses a gearhead, which ends with an offset drive shaft. The motor can be rotated in either direction, which moves the offset drive shaft closer to or further away from the axle, providing a method of increasing/decreasing tension in the timing belt. Despite the increasing capability of computers, it is always desirable to reduce the computational load of a processor. For this reason, it is highly desirable to find a way to control two

motors dynamically and leave the third fixed vs. running three simultaneously. To accomplish this the movements of the shoulder and elbow motors must be inertially isolated such that their movements minimally effect the base motor. This idea was incorporated into the design as shown in the image on the right side of Figure 7. The axis of rotation (marked by a line) for the base of the robot runs close to the rotation plane of links 1 and 2. Offsetting the cylinder upon which the upper links are attached decreases the moment arm to the base's axis of rotation, decreasing the effect of upper arm motion on the rotation of the base. The effects were not necessarily quantified and compared, but the general idea was implemented. The friction and effective inertia in the base motor has proven to be enough to prevent movement of the base during throwing motions.

Figure 7's left image also illustrates the use of the elbow motor as a counterweight, moving the center of gravity for Link 1 towards the shoulder's axis of

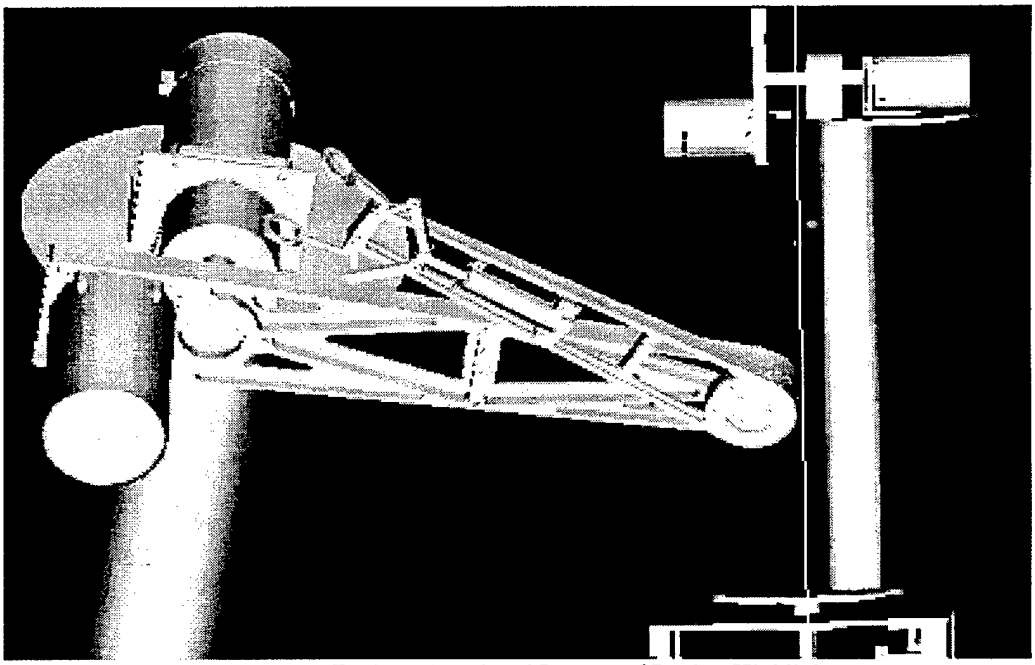


Figure 7 ProEngineer Rendered Images of Design Highlights

rotation, decreasing the moment of inertia. The placement of the motor further away from the elbow also allows the elbow joint to rotate fully without ever hitting itself as seen in the left image of Figure 7.

The initial model used for torque estimates assumed no friction. This of course is ideal and not realistic, the mathematical model requires another term for viscous friction and one for coulomb friction. The tedious part is finding the coefficients of friction to add to the torque calculations. To minimize friction as much as possible, dual bearings were built into the shoulder and elbow joints. The elbow joint turned out to have more friction than the shoulder joint. This is due to the shoulder motor bushing carrying no load while the elbow motor bushings carry the load of the elbow belt tension. Besides reducing friction, the bearings act as stress relief for the motor shaft.

Another stress relief feature of this robot was implemented after initial design, construction and assembly. It was desirable to use an acrylic tube. During the initial phases of the robot's use a crack developed on one of the bolt attachment sites in the plexiglass tube. This necessitated a redesign in order to avoid tube failure. The tube had been modeled as a whole and found to be strong enough to withstand the moments that it would experience, but no modeling of the attachment sites had been performed. The redesign included four ¼" steel rods placed parallel to the plexiglass tube with threaded ends. Once fastened with nuts, each rod was placed in tension, which compressed the tube, increasing its bending strength. After implementation, this redesign proved effective and successful in preventing further cracking and the eventually failure of the plexiglass tube.

The plexiglass tube and its reinforcements weren't the only alternative materials used. Aluminum is inherently weaker than steel and develops fatigue cracks faster than steel under the same loads. Out of all components, the base axle, shoulder and elbow shafts will experience the largest loads during operation. Therefore these components were constructed of steel instead of aluminum.

Another materials issue that developed was that of connection sites. Aluminum is much softer than steel and therefore wears more quickly than comparable steel. In the assembly of the robot some parts will rarely if ever be disassembled and other could be frequently assembled and disassembled. Such frequent use of steel bolts in aluminum threads could cause eventual failure. Even more extreme is the case of steel bolts in plexiglass, which is extremely soft relative to steel. For this reason, those connection sites that would experience frequent assembly and disassembly needed some sort of thread protection. Helicoils were chosen as the preferred method of protection. Helicoils are merely steel inserts that line a larger diameter threaded hole and provide a steel thread base for the bolts to connect to, reducing wear of the plexiglass and aluminum threads.

4.4 Robot Modeling

The precursory model used for sizing the motors was derived via the Lagrange-Euler method. The Newton-Euler recursive method as presented in John Hollerbach's course text [5] was used to derive this more specific model. This model estimates inertial, gravitational and frictional forces. The Newton-Euler Method is a recursive algorithm, which calculates the resultant torque on the end joint (or force in the case of a translational joint) due to forces and accelerations and then calculates the torque of the next joint and adds it to the torque of the end joint. This is repeated until all joints have

been accounted for and there is an estimated torque for each actuator. Equations (5) through (16) show the application of this method to a two-link planar pair robot. The base motor will be stationary during dynamic movement, allowing for use of a two degree of freedom dynamic model.

$$f_2 = m_2 g + f_{12} \tag{5}$$

$$f_2 = m_2 \ddot{r}_2 \tag{6}$$

$$f_1 = m_1 g + f_{01} - f_{12} \tag{7}$$

$$f_1 = m_1 \ddot{r}_1 \tag{8}$$

$$r_1 = cog_1 x_1 \tag{9}$$

$$r_2 = l_1 x_1 + cog_2 x_2 \tag{10}$$

$$n_2 = -cog_2 x_2 \times f_{12} + n_{12} \tag{11}$$

$$n_2 = I_2 \dot{\omega}_2 + \omega_2 \times I_2 \omega_2 \tag{12}$$

$$n_1 = -cog_1 x_1 \times f_{01} + cog_2 x_1 \times f_{12} + n_{01} - n_{12} \tag{13}$$

$$n_1 = I_1 \dot{\omega}_1 + \omega_1 \times I_1 \omega_1 \tag{14}$$

$$\tau_1 = z_0 \cdot n_{12} \tag{15}$$

$$\tau_2 = z_0 \cdot n_{01} \tag{16}$$

When evaluated, Equation (17) results, where cog_1 and cog_2 were replaced with their numerical values of 0.1135m and 0.08735m respectively.

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix} I_{133} + I_{233} + .0198m_2C_2 + .227m_2l_1 + .00763m_2 + .0874m_2l_1C_2 & .0198m_2C_2 + .00763m_2 + I_{233} \\ .00763m_2 + .0874m_2l_1C_2 + I_{233} & .00763m_2 + I_{233}\end{bmatrix}\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}$$
$$+\begin{bmatrix} m_2S_2(-.0198 + .0874l_1)\dot{\theta}_1^2 - .0397m_2S_2\dot{\theta}_1\dot{\theta}_2 - .0198m_2S_2\dot{\theta}_2^2 \\ .0874m_2l_1S_2\dot{\theta}_1^2\end{bmatrix}$$
$$+\begin{bmatrix} .114m_1gS_1 + .227m_2gS_1 + .0874m_2gS_{12} \\ .0874m_2gS_{12}\end{bmatrix} + \begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix} + \begin{bmatrix}\beta_1\dot{\theta}_1\\ \beta_2\dot{\theta}_2\end{bmatrix} \quad (17)$$

I - inertia tensor g - magnitude of gravity μ - coulomb friction

m - mass l - length β - viscous friction

Determination of real inertial characteristics can prove difficult and never result in anything but approximations, which can be tuned through empirical testing. The method used for finding better approximations of this robot's inertial characteristics was the mechanical properties option calculator in ProEngineer. The elbow motor's measured weight and calculated volume were combined to find an approximate average density, which was then applied in the arm assembly. Once the arm was virtually assembled the mechanical properties calculator determined the theoretical c.o.g.'s and inertia tensors for the virtual arm components. Although better, the estimate does not include the elbow motor's timing belt or the hand mechanism's spring or bowden cable.

4.5 Robot Parameters

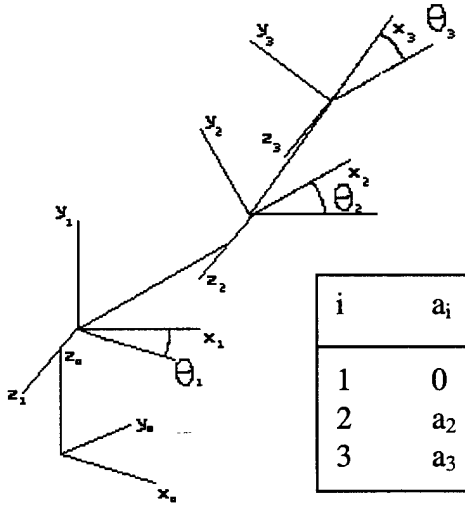
Figure 8 shows a Denavit-Hartenberg rendering of the robot. Origin 0 of the DH rendering is located in the middle of the rectangular base plate on the bottom. Therefore all measurements of distance should be calculated from the center of the base plate. The following table included in Figure 8 shows spatial and inertial parameters of the robot used in the kinematic and dynamic calculations.

4.6 Kinematic Equations

The Ball-Throwing Robot's planar pair kinematic and inverse kinematic equations are shown in Equations (18)through (23). A full derivation is included in Appendix B, which also includes a kinematic and inverse kinematic derivation including the third degree of freedom. The two degree of freedom kinematics structure is sufficient for the

a_1	0 m	a_2	0.02223 m	a_3	0.0381 m
d_1	0.9081 m	d_2	0.3048 m	d_3	0.2934 m
α_1	90°	α_2	0°	α_3	0°
$I_{1\,3,3}$	0.06147 kgm2	$I_{2\,3,3}$	0.002154 kgm2	m_1	1.932 kg
m_2	0.262 kg	Link 1 c.o.g.	0.01135 m	Link 2 c.o.g.	0.08735 m

Table 3 Current Values for D-H and Inertial Parameters



i	a_i	d_i	α_i
1	0	d_1	$\pi/2$
2	a_2	d_2	0
3	a_3	d_3	0

Figure 8 D-H Parameter Representation of Robot and Inertial Parameters Table

task of throwing, since the base motor only adjusts the direction thrown and is not calculated dynamically, but preset before the throw.[1]

Position:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 s_1 + \_ a_2 s_{12} \\ -a_1 c_1 - a_2 c_{12} \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} \theta_2 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \pm 2 \tan^{-1} \sqrt{\frac{(a_1+a_2)^2 - (x^2+y^2)}{(x^2+y^2)+(a_1-a_2)^2}} \\ \arctan(y,x) + \frac{\pi}{2} - \arctan 2\left(a_2 \sin\theta_2, a_1 + a_2 \cos\theta_2\right) \end{bmatrix} \tag{19}$$

Velocity:

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} &= \begin{bmatrix} a_1 c_1 \dot{\theta}_1 + a_2 c_{12} \dot{\theta}_1 + a_2 c_{12} \dot{\theta}_2 \\ a_1 s_1 \dot{\theta}_1 + a_2 s_{12} \dot{\theta}_1 + a_2 s_{12} \dot{\theta}_2 \end{bmatrix} \\ &= \begin{bmatrix} a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ a_1 s_1 + a_2 s_{12} & a_2 s_{12} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \\ &= \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \end{aligned} \tag{20}$$

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \frac{b_{22}}{b_{11}b_{22} - b_{12}b_{21}} & \frac{-b_{12}}{b_{11}b_{22} - b_{12}b_{21}} \\ \frac{-b_{21}}{b_{11}b_{22} - b_{12}b_{21}} & \frac{b_{11}}{b_{11}b_{22} - b_{12}b_{21}} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \tag{21}$$

Acceleration:

[1] For purposes of implementation, the zero angle position for the shoulder was redefined in all kinematics equations as straight down, such that calibrating and resetting the encoders would be easier.

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix}=\begin{bmatrix}-\left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 s_{12}-\dot{\theta}_1^2 a_1 s_1\\ \left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 c_{12}+\dot{\theta}_1^2 a_1 c_1\end{bmatrix}+\begin{bmatrix}a_1 c_1\ddot{\theta}_1+a_2 c_{12}\ddot{\theta}+a_2 c_{12}\ddot{\theta}_2\\ a_1 s_1\ddot{\theta}+a_2 s_{12}\ddot{\theta}+a_2 s_{12}\ddot{\theta}_2\end{bmatrix}$$

$$=\begin{bmatrix}-\left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 s_{12}-\dot{\theta}_1^2 a_1 s_1\\ \left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 c_{12}+\dot{\theta}_1^2 a_1 c_1\end{bmatrix}+\begin{bmatrix}a_1 c_1+a_2 c_{12} & a_2 c_{12}\\ a_1 s_1+a_2 s_{12} & a_2 s_{12}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} \qquad (22)$$

$$=\begin{bmatrix}-\left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 s_{12}-\dot{\theta}_1^2 a_1 s_1\\ \left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 c_{12}+\dot{\theta}_1^2 a_1 c_1\end{bmatrix}+\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}$$

$$\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}=\begin{bmatrix}\frac{b_{22}}{b_{11}b_{22}-b_{12}b_{21}} & \frac{-b_{12}}{b_{11}b_{22}-b_{12}b_{21}}\\ \frac{-b_{21}}{b_{11}b_{22}-b_{12}b_{21}} & \frac{b_{11}}{b_{11}b_{22}-b_{12}b_{21}}\end{bmatrix}\begin{bmatrix}\ddot{x}+\left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 s_{12}+\dot{\theta}_1^2 a_1 s_1\\ \ddot{y}-\left(\dot{\theta}_1+\dot{\theta}_2\right)^2 a_2 c_{12}-\dot{\theta}_1^2 a_1 c_1\end{bmatrix} \qquad (23)$$

During all path planning work the robot was always kept in the elbow down configuration and the robot was never commanded to pass through a singularity.

4.7 Hand design

The hand design experienced many revisions and stages of development. The driving requirements for the hand included: sufficient strength to hold the ball and a guarantee of no premature releases; fast enough opening dynamics to minimize residual effects on the ball's trajectory; and light weight, to minimize forearm inertia.

The ideal hand actuator would have the ability to control its position, the force applied to what it's gripping, have sufficient strength to hold the object secure throughout the throw (for a large rubber ball, estimated at 5.5 lbf worst case) and sufficient speed to release the ball consistently during successive throws. For purposes of this project sufficient accuracy is being able to throw into a garbage anywhere within the Throwing Arm's defined throwing range. Of these ideal characteristics, the project objectives made speed the highest priority and strength second. The motorized hand actuators considered

would have provided the position control, force control and strength, but would have failed to provide the estimated necessary speed response using motors within the assigned budget. This made another method necessary.

A solenoid from McMaster Carr proved capable of providing the necessary speed and strength. The position and force control are theoretically possible, but are beyond the objectives and scope of this project due to the complex nonlinearities involved in the problem. McMaster Carr provided specifications data, which included the solenoid force curve in Figure 9.

The graph shows force (lbf) vs. displacement (in) at 85% of its rated voltage (120 V)– full voltage translates the whole curve by adding approximately 3 lbf to the whole curve. This shows the greatest force at the beginning of the pull, which implies that the design should take advantage of the force spike. For this reason the solenoid was used in the design to release the ball, which would be held by a spring. This design uses the power surge, which occurs during the initial opening of the hand to expedite the opening motion, since the opening segment's speed is the most critical of the opening movement.

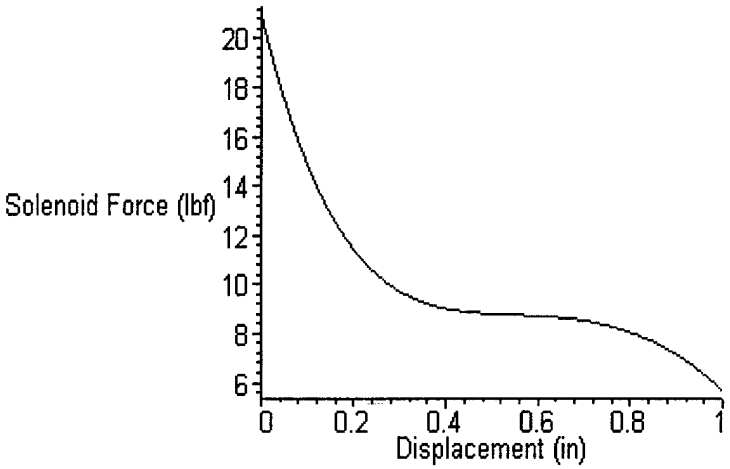


Figure 9 Solenoid Force Curve

Using a mass-spring damper system to model the hand's dynamics the ideal spring constant was calculated as shown in Figure 10. Ideally a spring could be found which would exert a constant force instead of a linearly increasing spring. Short of a constant force tape spring, the next best option was a long spring, such that the one inch of solenoid draw would minimally change the force value. The equation of motion for the system shown in Figure 10 is:

$$m\ddot{x}+b\dot{x}+kx=f(x)\Rightarrow \ddot{x}+\left(\frac{b}{m}\right)\dot{x}+\left(\frac{k}{m}\right)x=\frac{f(x)}{m} \tag{24}$$

$$\ddot{x}+2\zeta\omega_n\dot{x}+\omega_n^2 x=\frac{f(x)}{m} \tag{25}$$

$$\Rightarrow \omega_n=\sqrt{\frac{k}{m}} \tag{26}$$

In this specific instance $m=0.127kg+0.0283kg=0.1553kg$ and $k=594\,{}^{N}/_{m}$, which leads to the estimate of $\omega_n=\sqrt{\frac{k}{m}}=\sqrt{\frac{594}{0.1553}}=61.84\,{}^{r}/_{s}=9.84Hz$. From this the estimated period is 0.102s associated time required for release of the ball is 0.051s or one twentieth of a second.

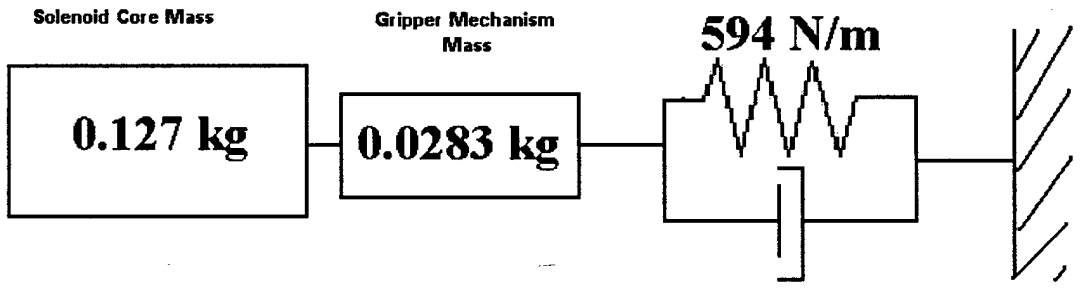


Figure 10 Spring Mass Damper Model of Hand

Although the solenoid had been chosen as the preferred method for moving the hand, the linkage had to comply with the solenoid's capabilities. The driving constraint of the solenoid came from its one inch of motion. This one inch of motion had to be converted into enough motion to move the hand's parts out of the ball's path, such that it could escape with minimal interference. Figure 11 shows the linkage chosen.

This linkage forces both "fingers" of the hand to moved synchronously. The movement was modeled mathematically in Maple to determine how the mechanical advantage varied through its range of motion. Furthermore it was necessary to verify that the single inch of solenoid motion would provide sufficient movement of the "finger" to accomplish the objectives. The optimal movement range was found to be 1.2 to 2.2 inches from the finger pivots.

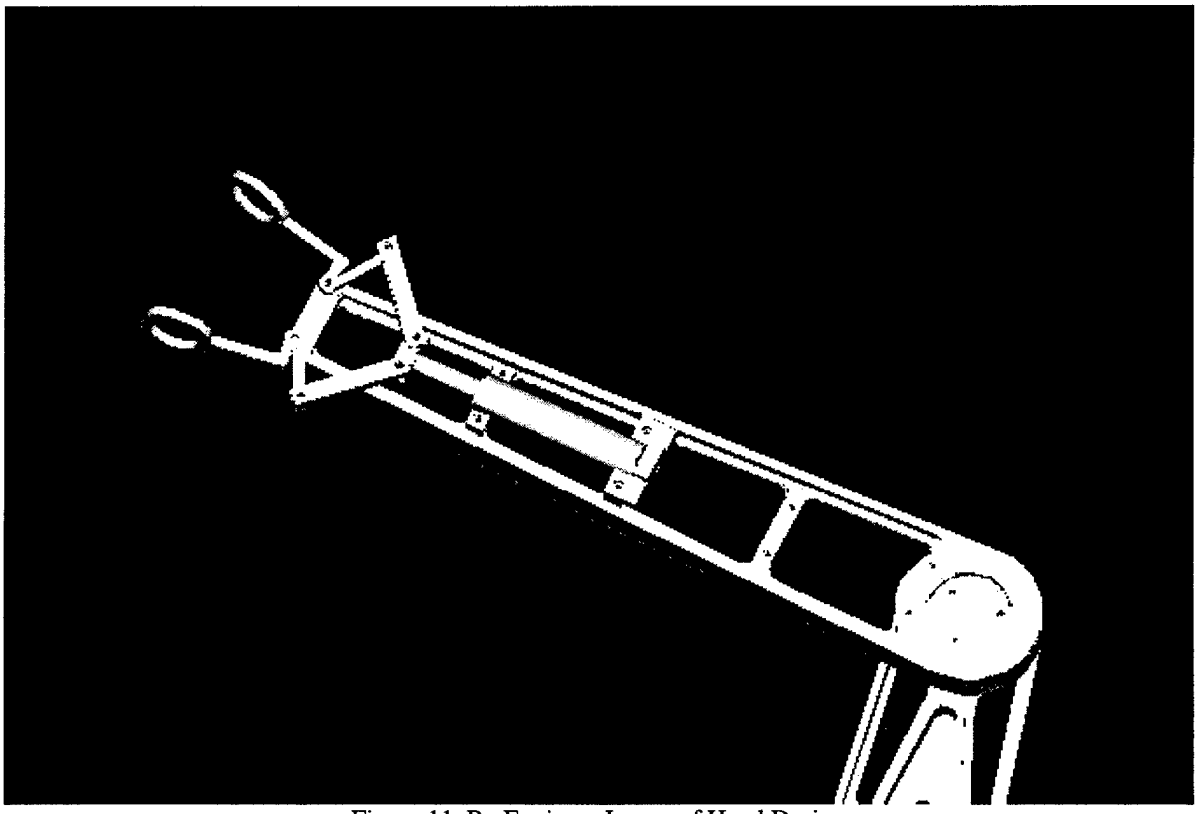


Figure 11 ProEngineer Image of Hand Design

The 5.5 lbf estimate of force to hold the ball in place is a worst case estimate and assumes no friction between the fingers and the ball. The applied force curve in Figure 12 can be seen to exceed the worst-case estimate for approximately 0.7 inches of the solenoid draw. This correlates to approximately 30 degrees of movement in each finger – full range of motion provides approximately 45 degrees of rotation in the "fingers" see Appendix C. Although this should be sufficient, if the actual forces do not approach the 5.5 lbf estimate, then the spring's preload could be reduced, such that the solenoid does not have to overcome such a large opposing force. Furthermore, if needed the solenoid can always be operated at its full rated voltage to generate even more force. In summary these preliminary sizing calculations showed that the solenoid design should indeed have the required speed and strength to meet the stated objectives.

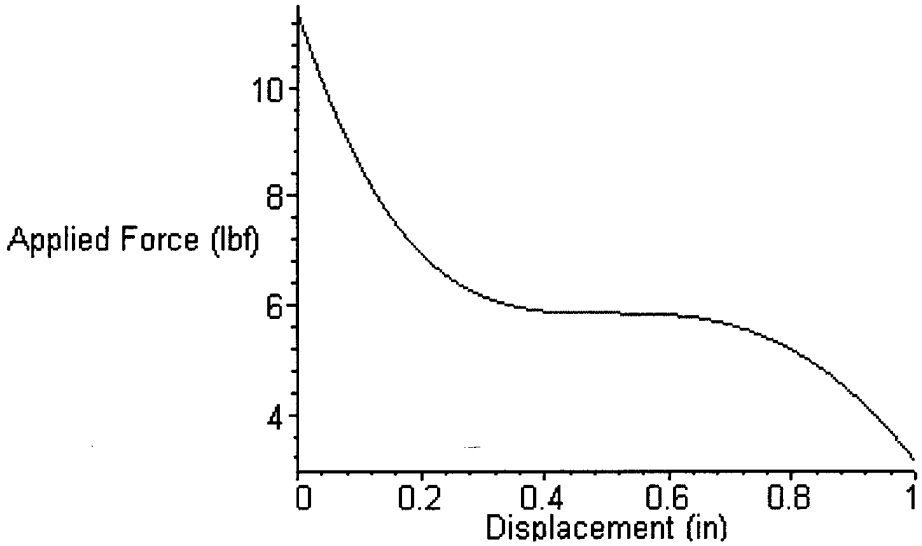


Figure 12 Available Force at Finger Tips

5 Robot Control

Construction and assembly of the robot are only the preliminary steps of the project. The challenging part of the project occurs in getting the robot to actually perform as intended. This happens via computerized control of the motors. In order to declare success in this aspect, the control algorithms implemented by the computer must be able to guide the arm through the motions, matching the ideal movements close enough to hit the targets with the rubber ball. This requires significant computational capability to read the present data sets, analyze them, calculate solutions and implement them hundreds of times per second.

5.1 Trajectory Planning

Two methods of trajectory planning were considered. The first method uses a cubic spline to interpolate between the start states, the release states and then another interpolation between the release states and the stop states. It treats each joint individually and does not account for interim end effector locations. This can lead to the forearm rotating more than 180 degrees from its initial position, which causes the ball to hit the upper arm. Two possible solutions to this problem include: 1) moving the elbow shaft out ¼" to circumvent the issue for 1.65" or smaller ball diameters; or 2) use end effector path planning and inverse kinematics to define the joint trajectories and never command the forearm more than 170 degrees in either direction.

An advantage to the first method is that the computer programming is much simpler. A general program which can account for all combinations of initial and final joint states is straightforward to write and debug. Such a program is then flexible enough to work with any input the students provide it. Furthermore, it's easier for a controller to

track linear acceleration signals, versus higher order acceleration curves. Although it does hold these advantages, a major disadvantage with respect to consistent throwing is the fact that the velocity vector is continually rotating throughout the whole throwing motion – the ball is therefore always being accelerated.

When using end effector path planning, the programmer can command the robot to follow a path of constant velocity for a brief period before, during and after ball release, allowing for moments of zero applied force from the arm. Challenges arise when writing a general program to account for all possible commanded throws. Software was written to this end, but a general solution was never thoroughly completed.

All tests were performed using the cubic spline interpolations for each joint individually. At this point, no testing of end effector path planning has been conducted. Without testing it's difficult to conclude with confidence whether or not the accuracy and repeatability would have changed. Although a period of zero acceleration during release would have been commanded, the question remains as to how well the controller could follow the commanded acceleration curves.

5.2 Feedforward Linearization

The Ball-Throwing Robot's control strategy uses a feed-forward term to cancel out the system's inherent nonlinearities, like gravity and asymmetric inertial properties. This implies the development of two key elements: 1) a more accurate mathematical model of the manipulator dynamics must be built to include coulomb friction, viscous friction, actual inertial properties and variable center of gravity locations and 2) a mathematical model of the motors and their response characteristics must be built, such that feedforward torques can be commanded via a voltage signal from the controlling

computer. The torque predictions could then be used to linearize the movement of the robot, such that a linear controller could then perform any error corrections using feedback gains based on linear control theories.

5.3 Actuator Modeling

Once the computer interface was built and assembled, it became relatively simple to collect simultaneous voltage, current, position and velocity data from the motors in various modes of operation. This data was necessary in order to build a mathematical model of the motor. The dSpace interface allowed the user to command various voltages while measuring the motor current and angular position of the motor shaft. Once this data has been collected, the torque constant, back-emf (ElectroMotive Force) constant and motor winding resistance can be extracted. Figure 13 and Equations (27) through (28) summarize the tree/cotree derivation of a motor's dynamic equations – the full derivation is in Appendix D. The dynamic equations describing the system can be simplified when the motor runs at a constant speed – all of the derivatives go to zero. This can be seen when Equation (29) is inserted into the electrical half of Equation (28) and Equation (30) results.

$$V_R + V_M + V_L = V(t) \qquad T_m + T_J + T_{B_v} + T_{B_c} = 0 \qquad (27)$$

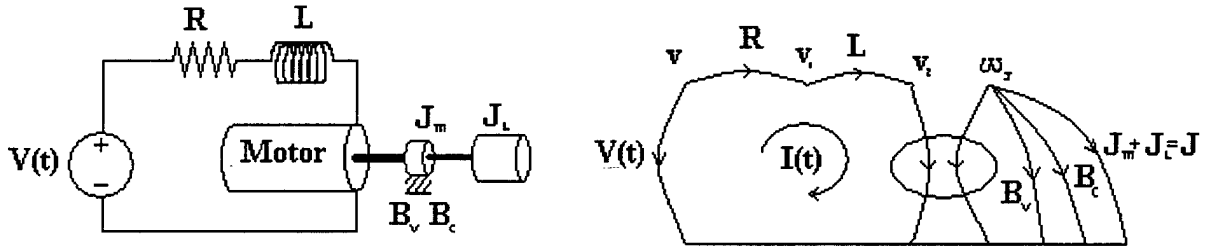


Figure 13 Tree/CoTree Motor Model

$$IR + k_e\omega + L\dot{I} = V(t) \qquad k_T I + J\dot{\omega}_J + B_v\omega_J + \mu = 0 \tag{28}$$

$$\text{at steady steady } \dot{I}=0 \tag{29}$$

$$\begin{aligned} &\therefore IR + k_e\omega = V \\ &\Rightarrow \tfrac{V}{\omega} = R\tfrac{I}{\omega} + k_e \end{aligned} \tag{30}$$

Although the motor does have inductive parameters, the dynamics due to inductance fade so quickly compared to the arm's movement that they were ignored in the modeling as suggested by [6]. So, when current, velocity and voltage information is available the back-emf voltage and characteristic resistance are readily obtainable.

Figure 14 shows a plot of data collected through the dSpace interface. All of these data points were collected during steady state operation. The linear regressions of both the forward and reverse data sets are displayed beside the plot. An average of the slopes was used as the motor's characteristic resistance and an average of the y intercepts

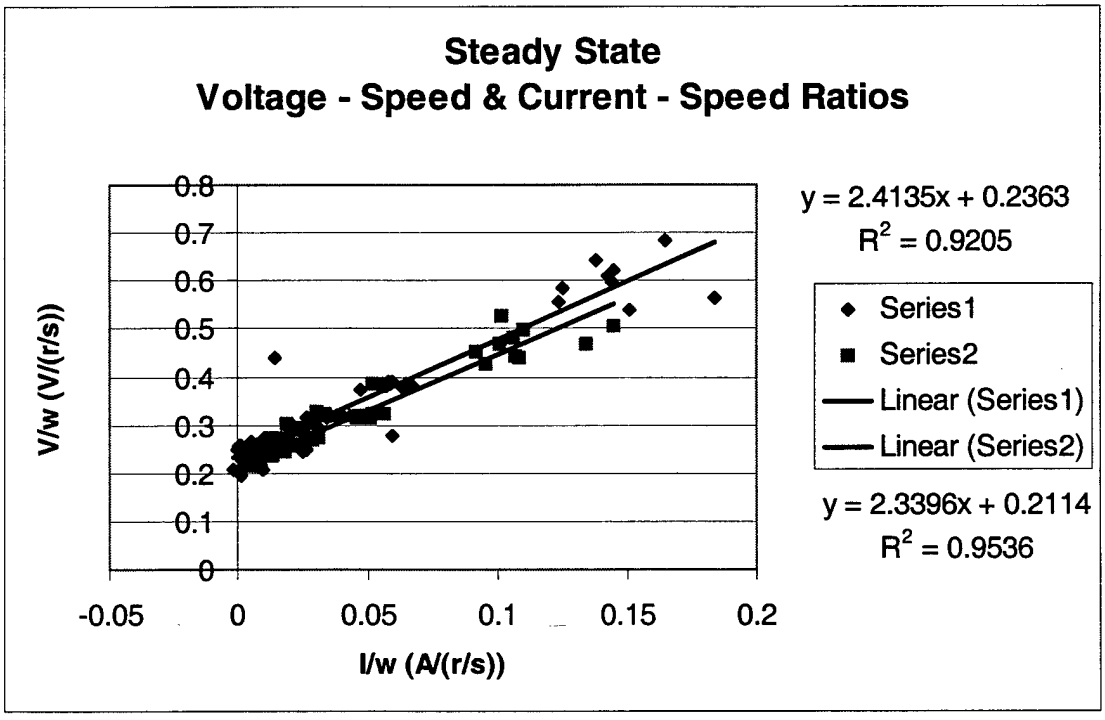


Figure 14 Data Used to Derive Motor Resistance and Back EMF Constant

was used as the motor's characteristic back-emf constant. Once these values are determined it opens the way to further motor characterization.

From the same data sets even more information can be extracted. Since all measurements were conducted in metric units the back-emf constant, $k_e = k_t$, the torque constant, relating current and torque. Equation (31) describes the mechanical side of the motor dynamics.

$$\begin{aligned} 0 &= k_T I + J\dot{\omega}_J + B\omega_J + \mu \\ &\text{at steady state } \dot{\omega}_J = 0 \\ &\therefore k_T I = -B\omega_J - \mu \end{aligned} \tag{31}$$

Since Torque = $k_T I$, once k_e is known, then the torque applied by the motor is known since the data set includes current readings. Figure 15 shows the same data expressed as torque vs. speed.

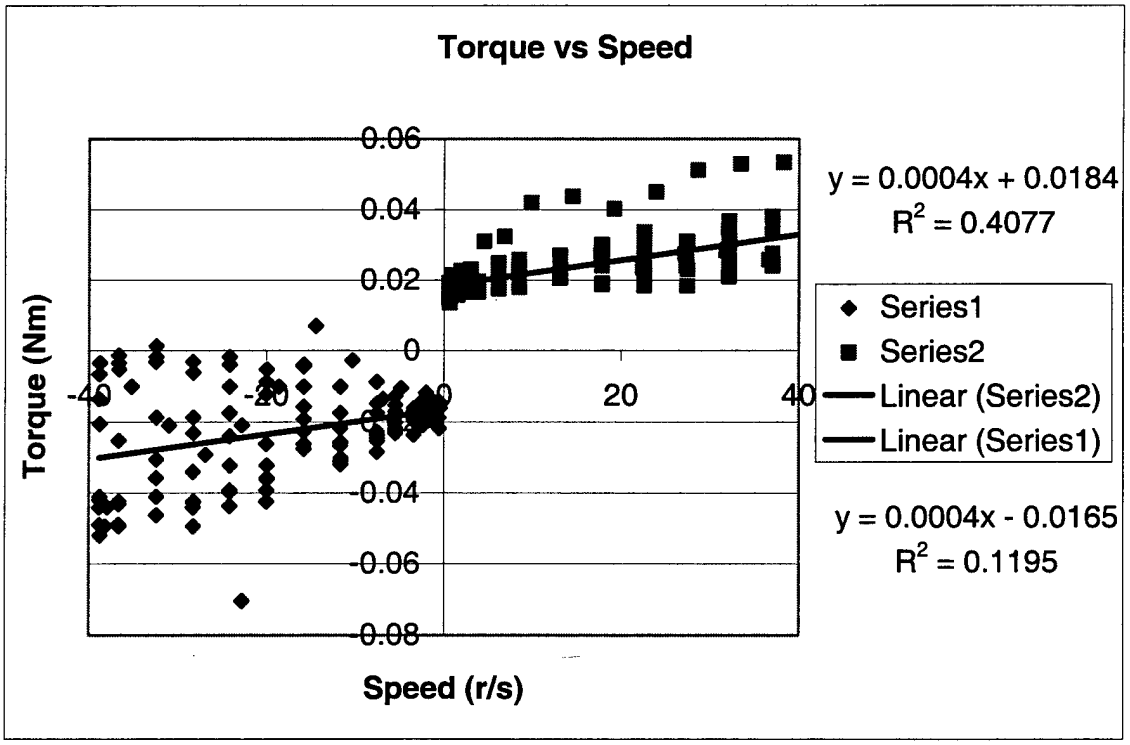


Figure 15 Data Used to Derive Friction Values

From Figure 15 values of viscous and coulomb friction can be calculated. In both the forward and reverse directions, there is an offset or base torque, which must always be overpowered in order for the motor to move. This is coulomb friction. Once the motor begins moving, then the torque needed to maintain it at certain speeds gradually increases as the speed increases. This torque that increases with speed is viscous friction. Each type of friction has been estimated in an attempt to provide the exact amounts of torque needed cancel out the nonlinear torque components. The viscous friction is approximately linear; therefore it is included in the linear model of the motor, and not in the nonlinear correction.

Figure 16 shows a signal flow diagram of the Ball-Throwing Robot's Joint 1. This diagram shows how the feed-forward torque is used to linearize the system, allowing the state space feedback gains to condition the motor according to linear control theories. Once the estimated torque and the corrective terms from the state feedback lines combine into the commanded torque, the inverse torque constant translates this value into a current, which is then translated into a voltage by using the empirically measured

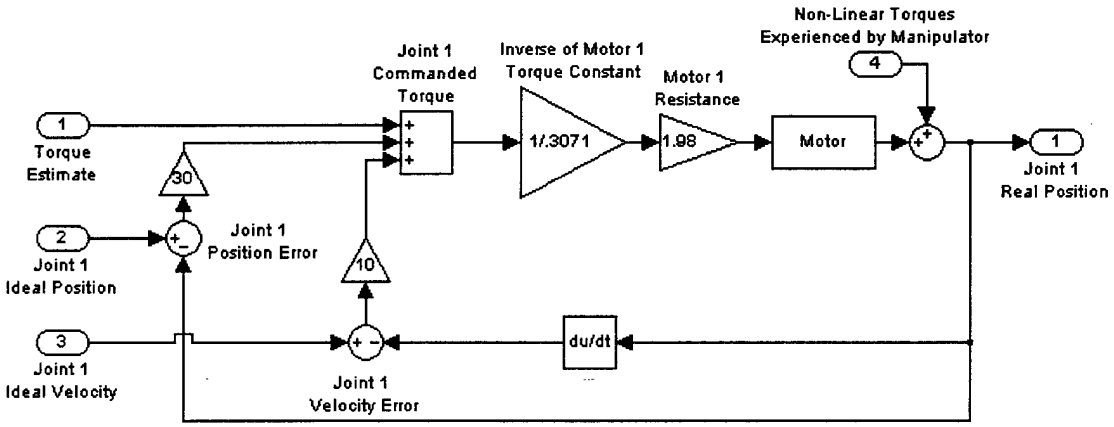


Figure 16 Signal Flow Diagram of Joint 1

resistance of the motor. This signal causes the appropriate movement in the motor if accurate nonlinear estimations and appropriate feedback gains are used.

At this point the desired torques are known and a schematic has been proposed, but motors use voltage or current inputs, not torque inputs. Before such a design can be implemented, the little black box called the motor must be modeled, such that the desired torque can be converted into a voltage or current to send to the motor. Appendix D contains a tree/cotree derivation of the state space motor model without inductance. This model provides the basis for all subsequent controller design. The equations of motion can only help as far as they use correct parameters. Values for the torque constant, k_t, and viscous friction, B, have already been approximated empirically via the test data introduced earlier in this section. That leaves the mass moment of inertia, J, yet undetermined for these motors.

The mass moment of inertia, J, can be extracted from step response data. When there is overshoot and stable oscillations decay, then the mass moment of inertia can be extracted if key values can be measured, namely time to peak, t_p, and percent overshoot, %OS. These two key data pieces allow for the calculation of the system's damping ratio, ζ, and natural frequency, ω_n, which provide direct correlations to the mass moment of inertia.

Similar to Equation (26) for a system with linear motion, Equation (34) shows the characteristic equation applied to rotational motion. When the empirically derived ζ and ω_n from the motors are entered into the equations below, then the only variable left unknown is J. The motor has no physical spring constant, K, just the effective spring of the unity feedback loop. Therefore the ω_n relation is less valuable, since it would require

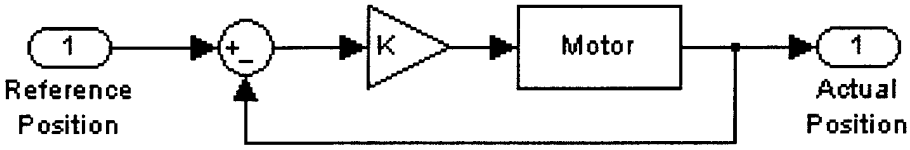
a theoretical estimation of the effective K and does not represent a physical component. In contrast, the viscous friction, B, has been measured in earlier analysis and can be combined with the measured ζ and ω_n to estimate the mass moment of inertia for the motor.

$$J\ddot{\theta} + B\dot{\theta} + K\theta = f(\theta) \Rightarrow \ddot{\theta} + \left(\frac{B}{J}\right)\dot{\theta} + \left(\frac{K}{J}\right) = \frac{f(\theta)}{J} \tag{32}$$

$$\ddot{\theta} + 2\zeta\omega_n\dot{\theta} + \omega_n = \frac{f(\theta)}{J} \tag{33}$$

$$\begin{aligned} &\Rightarrow \omega_n = \sqrt{\frac{K}{J}} \quad \& \quad \zeta = \frac{B}{2\omega_n J} \\ &\therefore J = \frac{B}{2\zeta\omega_n} \end{aligned} \tag{34}$$

Data for the shoulder and elbow motors was collected using a system like the one shown in Figure 17. The gain, K, was adjusted until there were a few oscillations, such



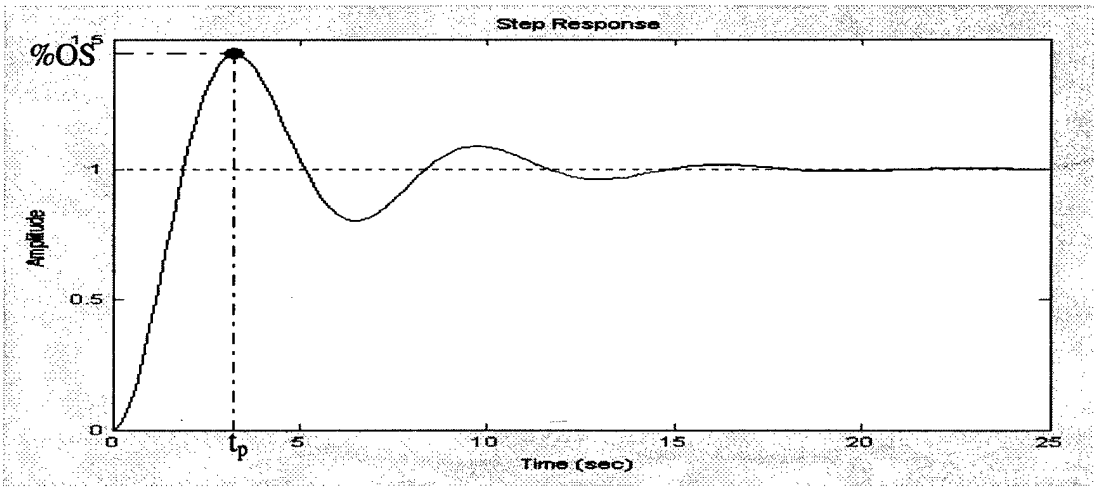


Figure 17 Data Collection Method for Motor Inertia Estimation

that the overshoot and time to peak were easy to measure. The adjustment of K is inconsequential for the purposes of this data. The adjustment of K changes the characteristics of the virtual spring in the system that's created by the unity feedback loop shown in Figure 17. It does not change the actual friction and mass moment of inertia of the physical system. Furthermore, when analyzed mathematically, the adjustment of K does not change the denominator of the open loop transfer function. Therefore it does not affect the estimate of the mass moment of inertia, which only appears in the denominator of the transfer function.

Eight step responses were analyzed for the shoulder motor and eight for the elbow motor, from which average values were calculated. The shoulder motor mass moment of inertia was not measured since it will not be moved during the throwing motions. The configuration at the time of the measurements was not conducive to the measurement of base motor mass moment of inertia. Furthermore the base motor's friction is sufficient to hold it steady during the throwing motions of the arm; therefore a precise dynamic model of the base motor configuration is not necessary. The base motor's mass moment of inertia is greater than the other two motor's mass moment of inertia, therefore an estimate of its value was made to use when needed, but the estimate is not based on data collected from the base motor. With defined motor parameters, all the information needed to derive a state space model is available for controller design.

5.4 Tracking Error Compensation

The feedforward linearization theoretically cancels out all of the manipulator nonlinearities, allowing for a linear controller to be implemented. Theoretically, the feedforward term can cancel out all nonlinearities. Then from the linear controller's

perspective, it appears to move only the motor, with no additional load attached to it. The moving position reference and velocity reference provide the input for the linear controller to follow. Although the feedforward term is calculated as accurately as possible, it can never be perfect in a real world, therefore errors will arise, which require a controller strong enough to correct for inherent modeling errors and disturbances.

State space methods were used to model and build the linear controller. This controller ideally needs to have the ability to correct for errors quickly and accurately. The resulting equations of motion derived in Appendix D are:

$$\begin{bmatrix} \dot{\theta}_J \\ \dot{\omega}_J \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} \theta_J \\ \omega_J \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_t}{J} \end{bmatrix} \ell_{in}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_J \\ \omega_J \end{bmatrix} \tag{35}$$

For purposes of discussion, let A, B and C relate to the matrices of Equation (35) as shown in Equation (36).

$$\dot{\overline{x}} = A\overline{x} + Bu$$
$$\overline{y} = C\overline{x} \tag{36}$$

where $\overline{x}$ is a vector of states, u is the input and $\overline{y}$ is a vector of outputs

Using full-state feedback, poles can theoretically be placed anywhere. This guarantees stability of the linear portion of the controller. The poles are placed in commanded positions by calculating the feedback gains for the states. Figure 16 shows a diagram of a motor with position and velocity feedback – full-state feedback in the case that inductance and amplifier filtering are ignored. When implemented the new system is:

$$\dot{\overline{x}} = \left(A - B\begin{bmatrix} k_1 & k_2 \end{bmatrix}\right)\overline{x} + B\overline{u}$$
$$\overline{y} = C\overline{x} \tag{37}$$

The only thing left is the choice of pole locations. The robot arm will have to follow a trajectory which will start with zero velocity and will accelerate smoothly to a release velocity and then decelerate again to zero velocity while returning to the start position. The micro controller will run at 750 Hz, updating the reference values each sample. Ideally the robot joints will execute a short period of constant angular velocities, such that the controller can converge on the ideal trajectory and hold it through the throw instead of forcing the robot to release the ball on a velocity trajectory spike. This brief release moment only lasts 0.05 s. The controller needs to correct for position and velocity reference changes fast enough to converge before 0.025 s, when the hand releases the ball.

In order to converge on this velocity and follow it, the compensated system should have a natural frequency twice as fast as this. This correlates to poles with a natural

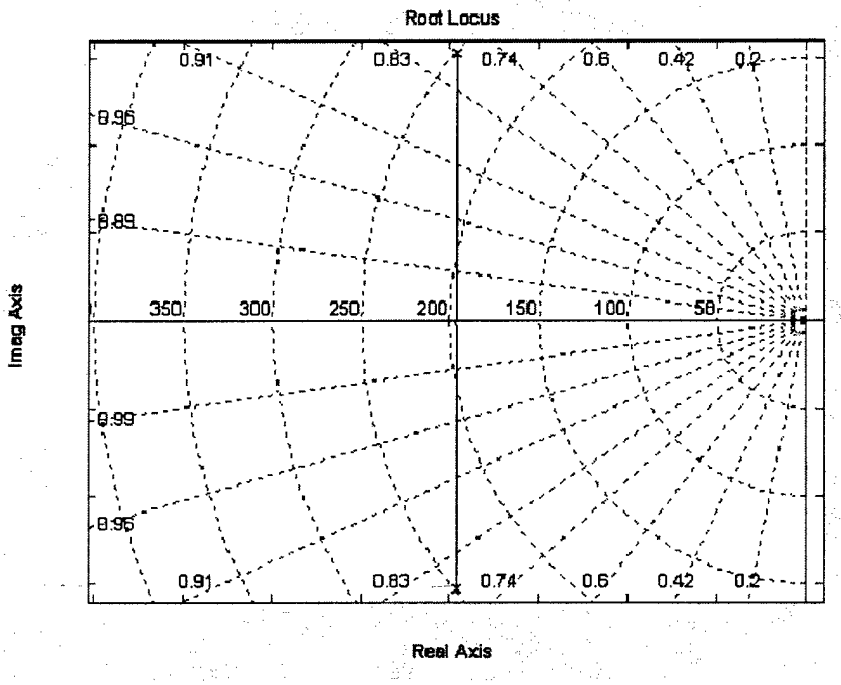


Figure 18 Pole Locations of State Feedback Compensated Linear Motor Model

frequency of 250 r/s. Figure 18 shows a root locus for the second order motor system. Poles with ζ of.766 and ω_n of 250 r/s as shown have associated feedback gains of -43.1 and -0.271for position and velocity respectively.

This root locus may be deceptive since it is based upon only two poles. This implies that the motor can never go unstable. A third pole actually exists due to inductance, maybe a fourth due to noise rejection filtering in the amplifiers. Therefore, the motor may become unstable with high gains. Most likely the amplifiers would saturate and the motor would just enter a stable limit cycle.

Through the combination of a feedforward term to cancel manipulator nonlinearities and a fast full-state controller to correct for errors while tracking position and velocity trajectories a controller was designed to accomplish the objectives of the ball-throwing arm project.

6 Robot Implementation

6.1 Safety Issues

The robot, when fully assembled, controlled and well behaved, should pose no threat to the students or operators. Reality always exposes new failure modes and design weaknesses. For this reason all potentially dangerous aspects of the robot must be considered and addressed. Especially since this robot will be taken to high schools to perform demonstrations for students. The chance of failure resulting in injury must be reduced to virtually zero.

6.1.1 Mechanical

The elbow and base motors are connected via timing belts, therefore slipping is possible if there is binding for any reason. The shoulder motor has a direct connection and will not slip without breaking part of the gear train or steel shaft. These motors were lab tested at 175 watts continuously by servo systems. The power source and amplifiers could source as much as 480 watts per motor, sufficient mechanical power to injure and maim a person, especially since the shoulder motor will not slip, it will either stall, break the robot structure or break the obstruction.

Certain precautions were built into the robot in order to avoid accident and injury. The first and foremost safety feature is one of distance. All operation of the robot will be performed outside its workspace. For those instances when the workspace may be approached or encroached by a person, another safety feature has been integrated.

The kill switch is simple and effective. It connects the capacitor output to the servo amplifiers' inputs. The kill switch can instantaneously interrupt power to the

motors of the robot or prevent it from beginning a movement. Although the kill switch disables the motors, it does not eliminate the danger of electrical shock, which is discussed in the next section.

Once the motors have been disabled the only motion not aborted by the kill switch is the solenoid, which operates independent of the power supply capacitor. This is acceptable considering the potential of an injury resulting from the motion of the hand assembly. There is only one potential pinch point in the hand assembly when the kill switch is off. It is shown within the circled area in Figure 19. When the solenoid plunger reaches the end of its one-inch pull it will exert no more than nine pounds of force between the pull rod and the attachment fixture. The solenoid has proven to startle people with its noise much more than with its actual movement or effected movement.

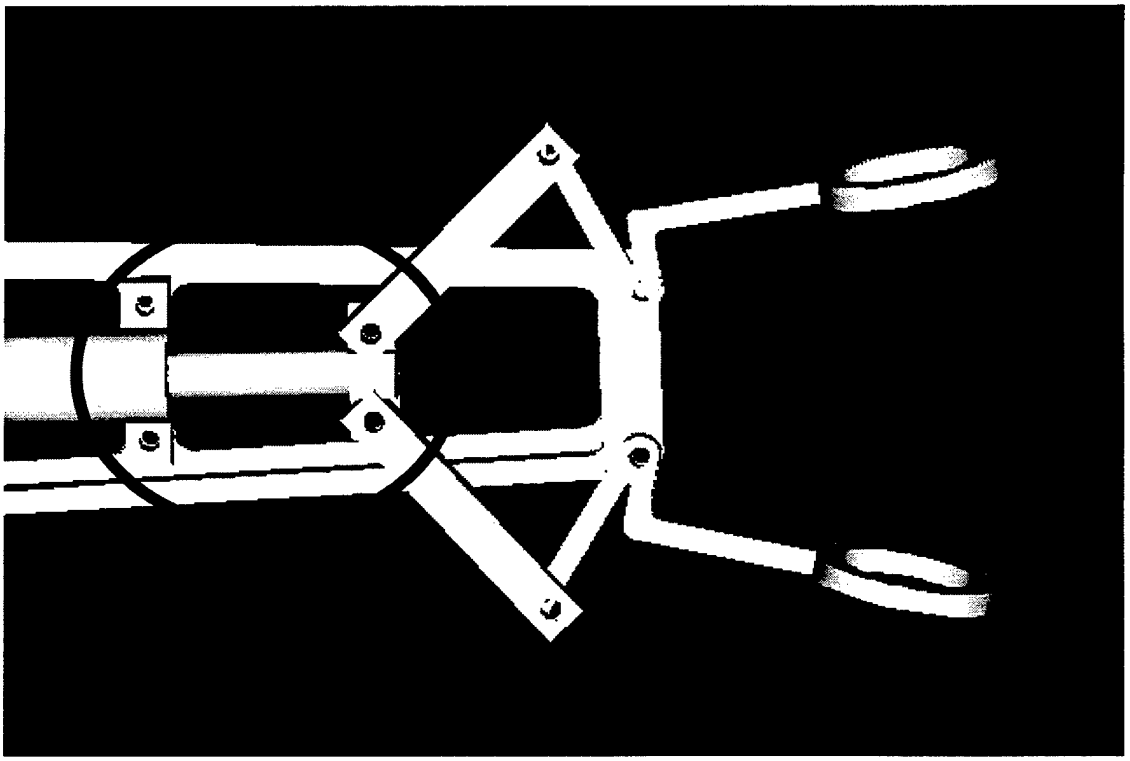


Figure 19 Pinch Point in Robot Hand

6.1.2 Electrical

The power source is connected directly to 120 VAC. The transformer connections are exposed within the robot base area. Any person working within the base area of the robot must know of this potential danger and take proper precautions. The power supply uses a full-bridge rectifier and a capacitor to reduce ripple. The capacitor is rated at 13mF – large enough to seriously injure and capable of killing a person in certain situations. Equation (38) calculates the amount of energy stored in this capacitor when charged.

$$U = \tfrac{1}{2}CV^2 = \tfrac{1}{2}(.024)(40)^2 = 19.2J \tag{38}$$

Currents of 5mA or less rarely cause damage, but 100mA for 1.3 seconds through the heart is sufficient current to kill a person. Such a current can be induced by 40V under the right conditions. Furthermore, this is a DC voltage, which penetrates skin and muscles more than 60 Hz current [7]. For this reason, the robot electrical system must be respected as such and should have a shield in place preventing students from touching dangerous components within the robot base.

6.2 Sensing Systems and Control Hardware

6.2.1 dSpace

The dSpace system is a DS1103 board and its associated software. The hardware includes a PowerPC 604e with a 400 MHz processor, 2 MByte local SRAM, 32- or 128-MByte global DRAM, 16 ADC channels, 8 DAC channels, 32 digital I/O channels, and many more features, which this project did not use. The dSpace board can take Matlab's Real Time Workshop Simulink models and implement them real time. It also allows for

the building of graphical user interfaces (GUI's) rapidly and easily. This made for a powerful interface of the computer and the robot hardware.

The Ball-Throwing Robot uses the DS1103 to collect precise position information, process the control algorithms and then output the control voltages to each individual motor. Each motor has an optical encoder mounted on it, which transmits quadrature signals to the processor through 15-pin subD connectors for decoding by the DS1103 processor. The robot currently uses encoder channels 4-6. The control algorithm is visually built and integrated in Matlab's simulink with Real Time Workshop blocks, then compiled and loaded onto the DS1103. Once loaded, it collects and manipulates data, outputting the resulting control signals via DAC channels 5-8. A program for characterizing the motors was developed which included the use of a ADC channel to measure the current monitor on the servo amplifiers.

6.2.2 Position Sensing

The sensing system for this robot is currently composed of encoders. The encoder sends out two streams of pulses. The pulse streams are always 90 degrees out of phase of one another as shown in Figure 20. The relative phase lead and lag of the pulses implies

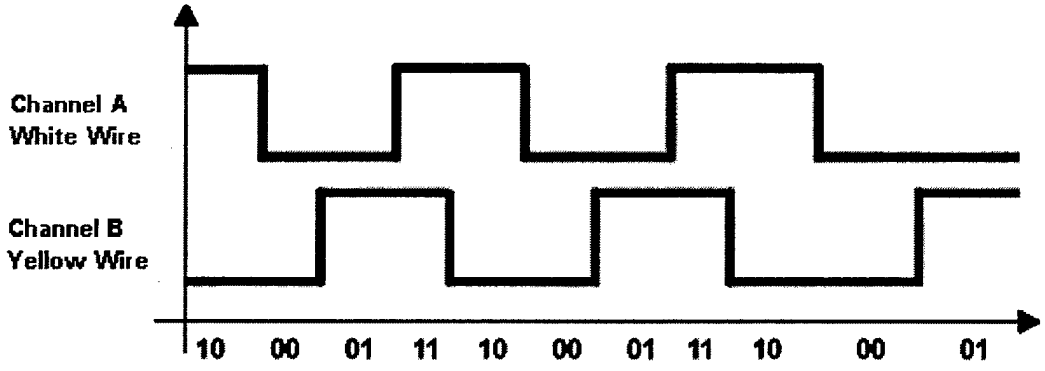


Figure 20 Quadrature Signal Decoding

the direction of movement. Each motor encoder has 100 pulses per revolution. There are two states per pulse, on and off. One pulse stream is sufficient to detect how far the motor has rotated, but cannot detect direction. Two pulse streams contain four possible states – 00, 01, 10, 11 - as illustrated in Figure 20. The algorithm receiving the data can compare the current state with the previous state to determine direction and counters can keep track of absolute position. The four state per pulse configuration allows for four discrete positions per pulse, which translates into 400 discrete positions per revolution and 18.5 * 400 => 7400 discrete positions per output shaft revolution. This accuracy is indeed greater than what is required for this robot's objectives.

Although all of the control is based upon the encoders, the design incorporated potentiometer verification of the encoder position, which would provide a method of auto-calibration too. Figure 21 shows a potentiometer attachment site. The current

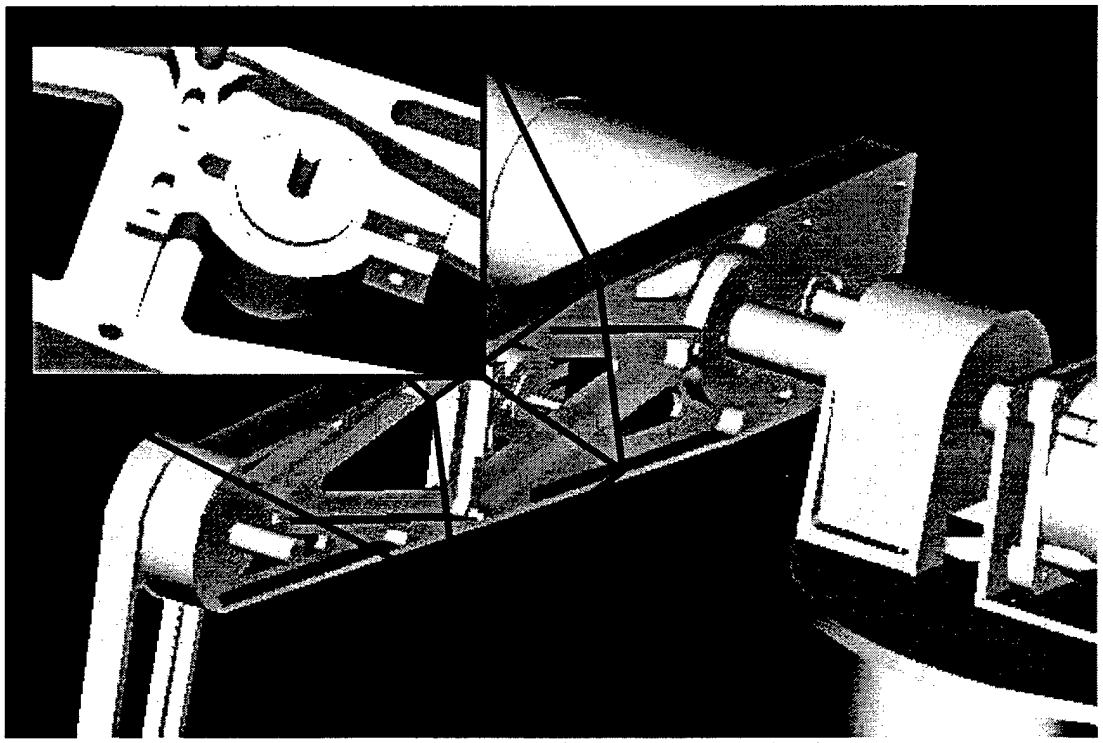


Figure 21 Potentiometer Option for Redundant Position Sensing

configuration of the robot does not include implementation of the potentiometers, just a design, which can incorporate potentiometer verification and calibration of the potentiometers. Implementation would require another three ADC channels and further computational support to account for the three potentiometers, but if desired they can be implemented.

6.2.3 Servo Amplifiers

Another integral piece of the control hardware is the servo amplifier. As earlier stated, this robot uses three Servo Systems Pulse Width Modulated (PWM) 12A8 servo amplifiers. These have the ability to operate in a current following or voltage following mode, sourcing 6A continuously and 12A peak at 20V-80V, with a switching frequency of 36kHz and operational bandwidth of 2.5 kHz depending on the applied load. It has adjustable: loop gain; current limit; reference gain; and offset/test. These are just a few of the specifications most applicable to this application. The amplifiers were adjusted to have zero offset and a reference gain of four.

The reference gain of four arises from dSpace's voltage output limitations. The dSpace DAC converters are capable of up to 10 volts out. The power supply provides up to 40 volts, thus a gain of four was used to allow for the full range of voltage application. Another non-intuitive gain is a dSpace gain. The voltage commanded in the GUI is multiplied by 10 at the DAC voltage out.

6.3 Micro Controller

The dSpace DS1103, although well adapted for this project, easy to use for implementing new programs and much faster than needed for this project, is not

particularly portable. It operates out of a desktop PC and has many large cables associated with it. For this reason, from the very beginning the project included the idea of a single board computer controller. Such a computer would provide a portable controller and user interface, through which the students could command the robot through various movements.

After two and a half months of shopping and research the Z-World PK2600 was chosen. The PK2600 is a controller unit, which includes two 19 MHz Z180 Zilog processors. One processor controls a ¼ VGA touch screen monochrome interface and the second processor controls the I/O ports. The display processor runs all of the user interface programs, introducing the project, displaying menus and taking the data needed to command the robot. The display then ports the information to the controller processor via an internal RS-232 serial link. The information can be used to precalculate the torque trajectories for the desired robot arm motion. When ready, an execute button on the display can be depressed, sending the command signal to the controller processor to start the throwing motion. The user interface and internal communication routines are still under construction by Roy Merrell and Danny Blanchard.

The controller has 32 lines of digital I/O. These lines can be configured in any of the following combinations: 8 in / 24 out; 16 in / 16 out; or 24 in / 8 out. Either 16 or 24 lines of digital inputs are used to read in two motor encoder signals simultaneously. Due to the relatively slow speed of this processor, decoding the quadrature signals in the software algorithm could prove burdensome and significantly decrease the sampling rate of the controller. Agilent Technologies donated external quadrature decoders to solve

this problem[2]. Although dSpace distinguishes quarter pulse positions, the Agilent quadrature decoders only resolve whole pulses. These quadrature decoders count the quadrature pulses, measure the phase relationship to determine the direction and magnitude of the movement and output the position via digital lines. These digital signals are read by the controller to determine position and velocity for use in the control algorithm. The controller calculates the outputs and has 7 PWM analog channels to use to send the signals to the amplifiers.

The quadrature chips have various modes of operation: 8-bit; 12-bit; and 16-bit counters. Each obviously has its advantages. The 16-bit counter is normally used for absolute position monitoring and the 8-bit has options adapted specifically to incremental operation. The robotic arm was originally designed to channel two streams of 8-bit position information, one from each quadrature chip to the controller through a single 16 line digital input port. This method was encountering drift errors during operation.

The incremental operation mode requires a reset signal from the controller immediately after a measurement is taken to reset the counter, such that overflow between samples is avoided. Although the source and nature of this drift is yet unexplained, a possible explanation results from the speed of the controller. One line of code measures the current position and the counter is not reset until the next line of code executes. Any movement between the measurement of position and the reset command would be lost entirely.

One possible remedy would involve a different digital I/O port configuration. Currently there are 16 digital lines in and 16 digital lines out. If the configuration were changed to 24 lines in and 8 lines out, then each motor could use the quadrature decoder

[2] Data Sheets included in Appendix I

counter as an absolute position reference in stead of as an incremental position reference. This would introduce another level of complexity.

The quadrature decoders each have only 8 lines of digital output, therefore the 12 and 16 bit counters must be serially output 8 bits at a time. This would require more code to read the position. The controller would then use one line of code to read the Motor 1 position and the next line to read Motor 2. This would require a different circuit design which would use a controller commanded signal to regulate which quadrature chip's signal was passing to the input port at which time. This would allow the controller to latch the data for read processes and not lose data due to movement during the read period. The quadrature decoders continue real-time position monitoring in the background even when data has been latched to the output ports, therefore drift would not arise from slow controller coordination.

6.4 Control System Prototyping

The controller described in Section 5 was conceptually built and simulated early in the robot's development. When implemented and tested the results were not fully satisfactory. The variance in the robot's throwing performance was too sporadic. The robot's hit spread at a distance of 12' was larger than the lab garbage can - about 14", which is unsatisfactory. The robot should be able to consistently throw into an area the size of a garbage can and ideally much smaller. This way, the robot could compete against a student for accuracy and consistency and win. For this reason, the controller had to be modified.

Various things like empirical gain adjustment, different feedback arrangements and model-based control were tried. The initial position and velocity gains were adjusted and

empirically tuned. The result was an aggressive controller with increased consistency, but still not good enough.

The next variation tried was acceleration error feedback. When attempting to speed up the tracking response of the original controller, velocity oscillations developed as the controller tried to converge on the velocity trajectory. Velocity feedback is commonly used to damp out oscillations in a position control system. Similarly it made sense that acceleration feedback could feasibly damp out velocity oscillations. So it was added. The change was dramatic. The velocity oscillations disappeared and the system tracked both position and velocity profiles better than ever before. The tough question though was what gain to assign to the acceleration feedback. The gain implemented was tuned through multiple trials. Another side effect of the acceleration feedback used proved to be a safety improvement for the system.

Previous controllers experienced quick jerks when large step disturbances were encountered. For example if the arm was jarred to a nonzero position while the power was off and the power was subsequently applied, then the links would quickly jerk back to position, sometimes experiencing limit cycles around their desired position. The flailing links could potentially injure a person if in the way.

Part of the reason for this behavior is because the system is not feedback linearized, but feedforward linearized and when the robot's feedforward trajectories are not streaming into the controller, then the linear controller is attempting to control a very large nonlinear inertia for which it's not designed. When the acceleration gain was added, it killed the limit cycle tendencies because the post throw acceleration and velocity references are zero they tame the position feedback signal, such that the links

creep back to the desired position at very slow rates, taking sometimes five second to move back to the desired position. Such a slow speed does not endanger anyone, but when used to track trajectories there is no velocity oscillation and the results were the most consistent of any controller used thus far. A challenge resulting from such implementation is that no theory found explains how to choose an acceleration feedback gain. The closest method found is model-based control.

Model-based control attempts to make the motor appear like a block with unit inertia to the controller. When configured for tracking a trajectory, the acceleration term is added directly into the control signal as a feedforward term as shown in Figure 22. This was attempted in parallel with the feedforward linearization developed earlier. Equations (39)-(44) derive the idea behind this method as presented by [6]. They show that k_p and k_v can be chosen to command any error response desired, assuming that the system is correctly modeled.

$$m\ddot{x} + b\dot{x} + kx = f \tag{39}$$

$$f = \alpha f' + \beta \tag{40}$$

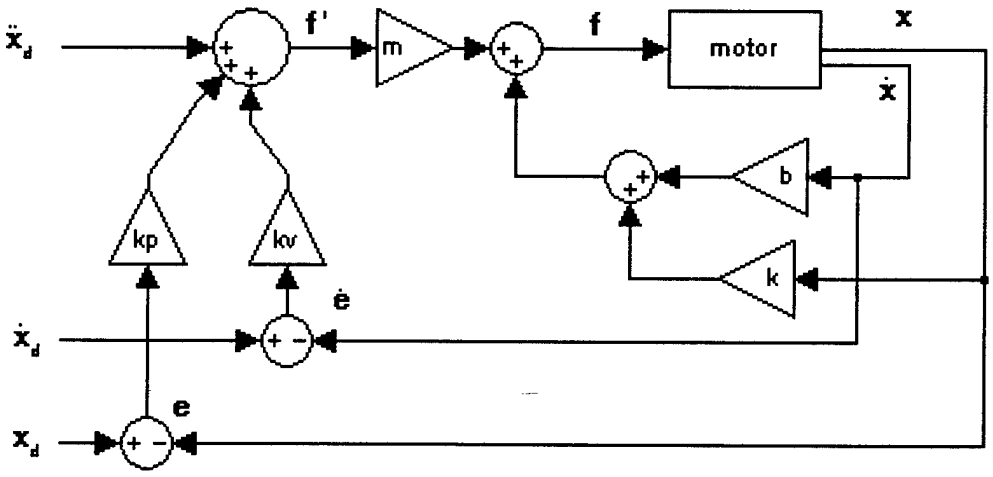


Figure 22 Model-Based Controller Signal Flow Diagram

Define:

$$\begin{aligned} \alpha &= m \\ \beta &= b\dot{x} + kx \end{aligned} \tag{41}$$

Insert Equation (41) into (40) and then (40) into (39) to get:

$$\ddot{x} = f' \tag{42}$$

From Figure 22:

$$f' = \ddot{x}_d + k_v\left(\dot{x}_d - \dot{x}\right) + k_p\left(x_d - x\right) = \ddot{x}_d + k_v\dot{e} + k_p e \tag{43}$$

Combine (42) and (43) to get:

$$\ddot{x}_d - \ddot{x} + k_v\dot{e} + k_p e = \ddot{e}_d + k_v\dot{e} + k_p e = 0 \tag{44}$$

This shows that the choice of position and velocity feedback gains can create an error signal that always decays to zero. A critically damped response occurs when $k_v = 2\sqrt{k_p}$. These two controllers were implemented along with the original controller. These two subsequent controllers proved much more effective than that in the original design.

7 Testing and Performance

7.1 Operating Procedures

Appendix E contains a set of dSpace operating procedures as well as the code used throughout the controller with explanatory comments. These instructions will lead a user through all the startup steps involved in getting the robot ready to throw.

Once all initialization and setup is complete, then the dSpace graphical user interface is the main portal. All parameters except link centers of gravity can be adjusted through the GUI. This allows for fine-tuning on the fly. Once all parameters have been

chosen, the power button may be depressed, connecting the capacitor to the power amplifiers. At this point, the Move button should be depressed once to adjust the base motor angle then the Start button should be depressed to execute the throwing sequence. Once the arm has thrown and moved back to its initial position, then the power button should be released, making the arm perfectly safe to approach.

7.2 Experimental Setup

All throwing experiments were performed in MEB 2178. Figure 23 shows how the robot sits on a cabinet in the office, from which it throws. A pillow of packing bubbles caught the ball and the distance thrown was recorded.

7.3 Results

Many test throws were performed with the intent of determining the Throwing Arm's accuracy and repeatability – 299 of these throw distances were recorded. Appendix F contains tables summarizing all test data. Tools used to collect informative

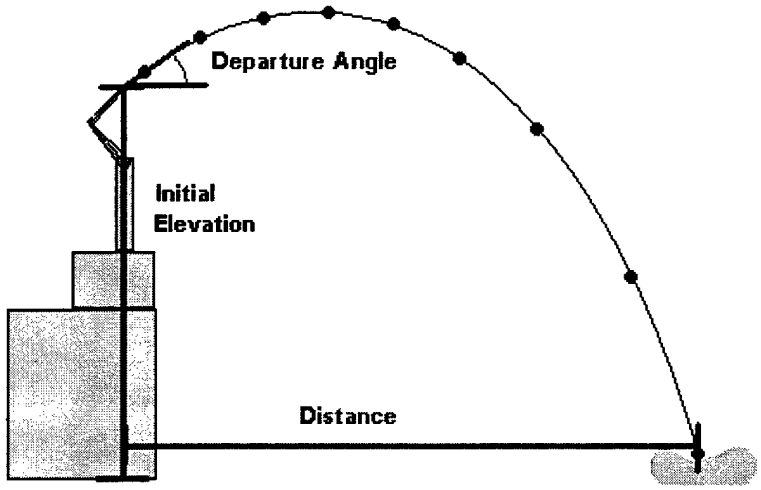


Figure 23 Experimental Setup

data included telemetry via the dSpace interface, a digital video camera and a standard tape measure. For each combination of release velocities, multiple throws were observed and recorded. Those results were averaged to find average throwing distances for each setting. The error between the predicted distance and the average throw distance and the standard deviation of the actual throw distances was calculated. The data presented compares the two most successful controller designs implemented thus far, the model-based controller and the acceleration feedback controller.

The model-based controller's accuracy grew continually worse, the farther it tried to throw. Its decrease in accuracy declined at a faster rate than the acceleration feedback controller. Figure 24 shows the predicted distance on the x-axis. The data points represent the error between the average throw distance and the predicted distance and the standard deviation of the throw distances. For the model-based controller, both the consistency and the accuracy of the robot at each setting grew continually worse as the

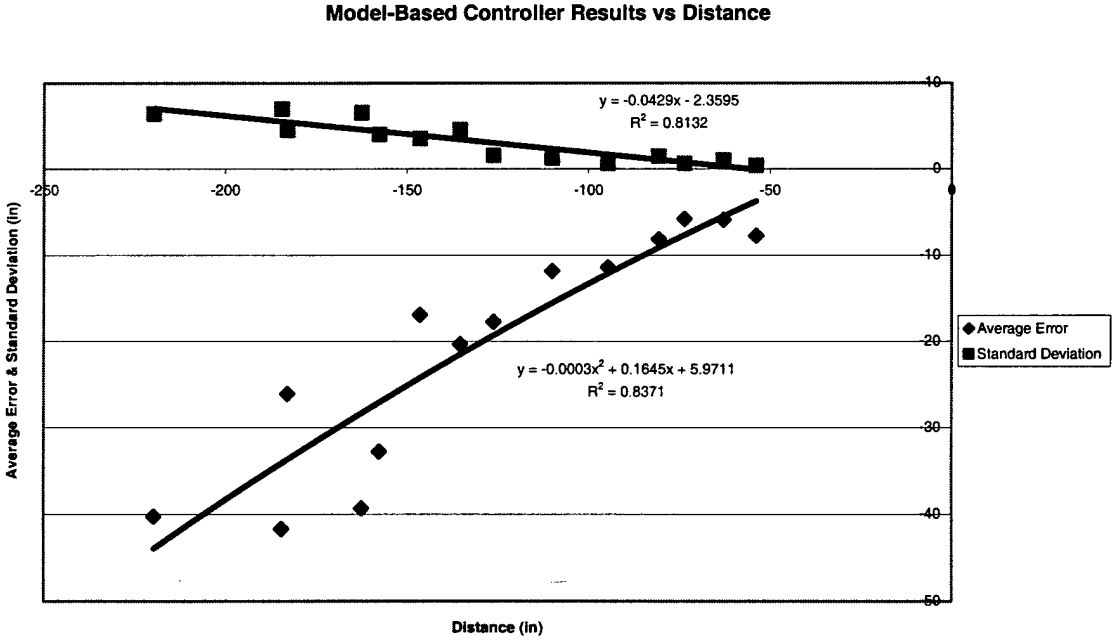


Figure 24 Test Data from Model-Based Controller Throws

distance increased. The consistency declines linearly, while accuracy's decline appears parabolic.

The acceleration feedback controller test data is different for the same commanded trajectory profiles. Figure 25 shows the results in the same format as the model-based controller data in Figure 24. The acceleration feedback controller's standard deviation also changes in an approximately linear fashion, getting worse the farther the robot throws, but the slope of its trend line is smaller than the model-based controller's. Furthermore the consistency of the acceleration feedback controller declines at a slower rate than the model-based controller.

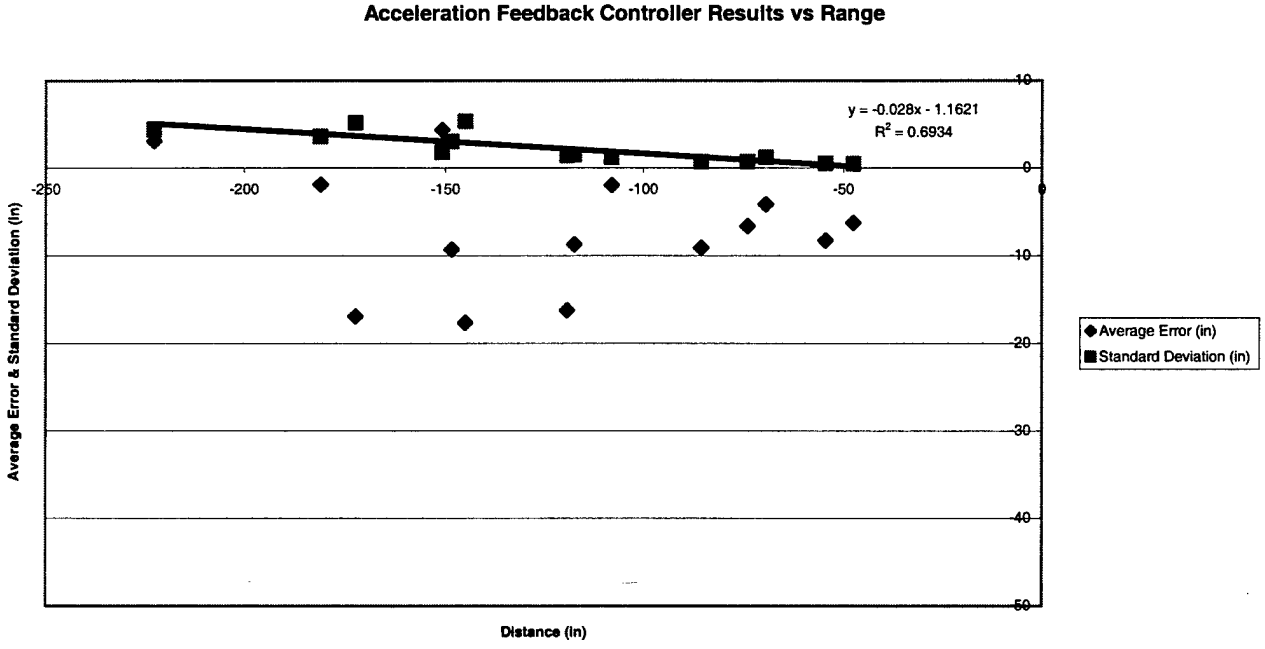


Figure 25 Test Data from Acceleration Feedback Controller Throws

Although the model-based controller's error seemed to grow parabolicly, the acceleration feedback controller does not seem to have an error that is an explicit function of distance. Instead further analysis of the data suggests that the error is a function of the commanded release angle and distance. For near zero release angles, the error was a function of distance similar to the model-based data in that the accuracy's decline could be approximated parabolicly. Figure 26 shows the data collected during near zero release angles throws. The 10 to 45 degree release angle throws show a very different trend. There is no particular relation between release angle and error, but as a group the error is quite small compared to the model-based controller. For the same commanded trajectories, the model-based controller had only three throws with less than 10 inches of error, the acceleration feedback controller in contrast had only three throws with more than 10 inches of error. Common to all is the steady increase of the standard deviation. Out of all data, the throws released at angles of 10-45 degrees have the smallest standard

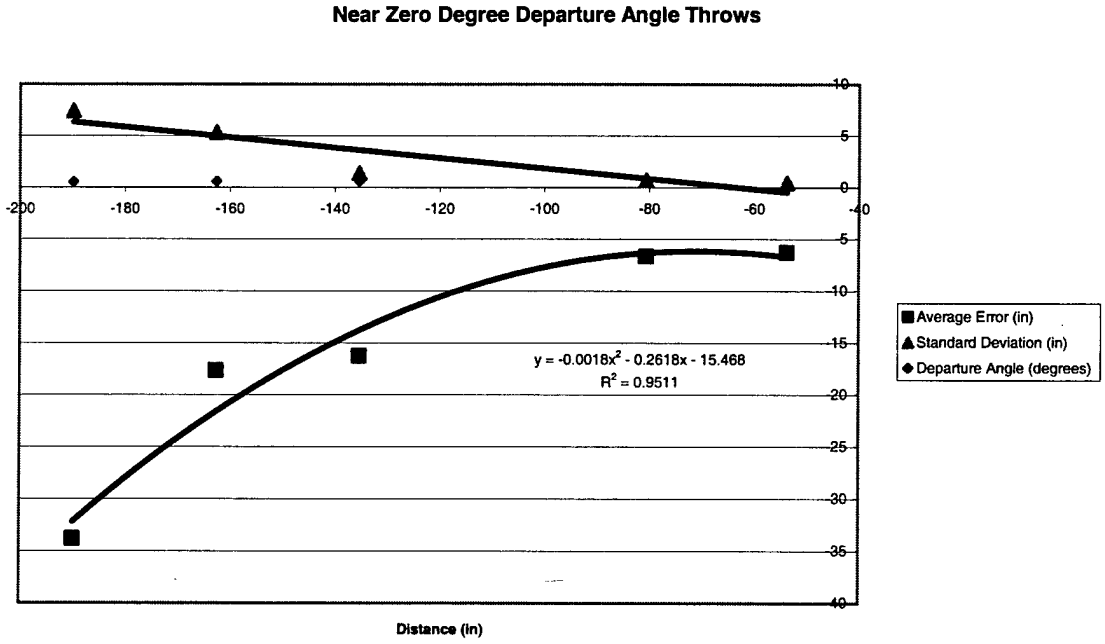


Figure 26 Acceleration Feedback Throws with Near Zero Departure Angles

deviation slope of any.

The next question asks for an explanation of why the model-based controller does not have the accuracy or the consistency of the acceleration feedback controller. The model-based controller relies upon accurate modeling of the motors and their loads.

Figure 15 on page 36 shows some of the data from which the motor model parameters were derived. Figure 15's data does not fit any line very closely. When averaged it reflects a general slope, but is far from the ideal that theory predicts. All data collected from these motors was just as noisy. The parameters derived from this data may be flawed. Furthermore, all of these parameters are characteristic parameters used to linearize nature, which is essentially non-linear. They are all approximations that help us try to predict nature. Another tool provides more insight into this question.

The dSpace data capture illuminates what's happening inside each controller. Appendix G overlays the responses of both controllers to the same input trajectories. The

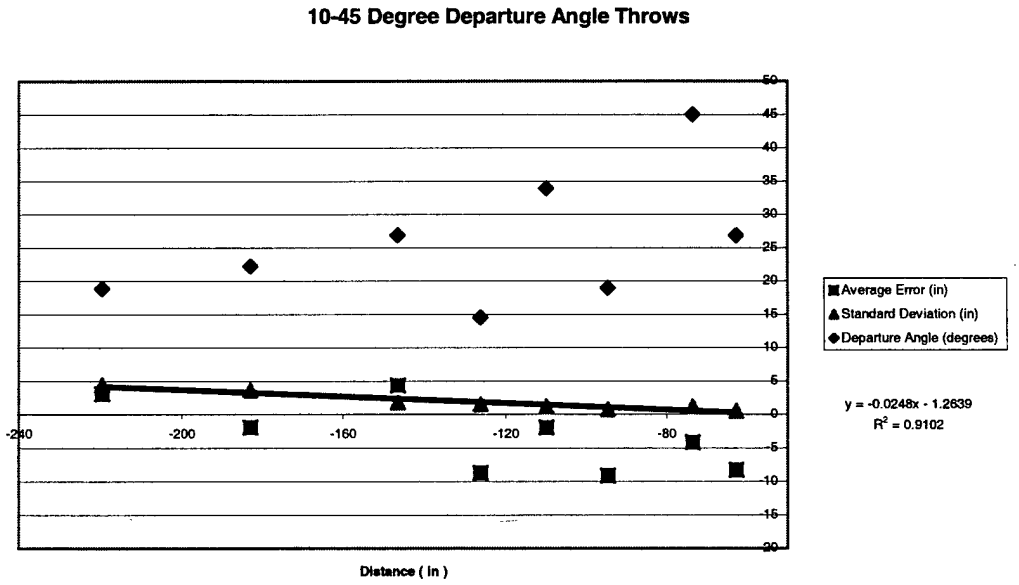


Figure 27 Acceleration Feedback Throws with 10-45 Degree Departure Angles

model-based controller velocity signal has oscillations and drops off quickly at the points of sudden acceleration changes. In contrast the acceleration feedback has virtually no velocity oscillations and holds the zero angular acceleration regions better than the model-based controller. The acceleration feedback controller holds its velocities relatively constant through the ball release segment and matches the desired velocity closer than the model-based controller. The model-based controller has oscillatory peaks that can vary in magnitude and timing. This could explain the model-based controller's generally lower accuracy and its lower consistency.

One characteristic common to both controllers is a slight lag in the velocity tracking, most noticeable in the shoulder velocity profiles. Both break at the instant the acceleration goes to zero and both have approximately the same lag. When implementing the velocity feedback, the incremental encoder's position signal was filtered and differentiated. The filter is second order with a break frequency of 70 r/s, so the velocity signal is delayed before it enters the control, which has its own natural response delay. This may be an evidence of those delays in the system.

An attempt was made to observe the interaction that occurs during the release of the ball, to see how much the fingers disturb the trajectory of the ball during release. A digital video camera with the capability of capturing 30 frames/sec was used to film the arm during its throwing motion. Appendix H contains a selection of the 60 or more frames taken during the arm's throwing motion. These focus mostly around the release phase. Although the camera takes 30 frames per second, the arm is still moving fast enough to blur and no specific information about the interaction between hand and ball

can be derived from the pictures, at most an estimated departure angle from the ball's fairly linear blur.

8 Conclusions and Future Work

A Ball-Throwing Robot capable of throwing balls on command to designated points in the room has been successfully built and demonstrated. The accuracy and repeatability of the robot vary with distance and departure angle, but have shown that departure angles from 10 to 45 degrees have the best accuracy and repeatability

More can be done to improve the robot's capabilities. Further investigation of the interaction between the hand and ball could possibly further explain accuracy variance. In order to accomplish this a high speed camera would be needed. Such a camera would allow for observation of the hand's affect on the ball's trajectory during the release motion. This information is key to determining exactly what is happening in the throwing motion. For this reason evidence and possible explanations are presented, but further work is needed to fully understand and hence fully control the accuracy of this robotic arm.

Another issue worthy of further work and development is the end-effector path planning. This method of throwing could potentially be much more repeatable and accurate than the method current used.

Mechanical design issues that could improve the robot include:

1) Modifying the solenoid attachment point. It currently forces the cable to make some awkward curves. A straightened cable could decrease the solenoid response time, allowing for more precise ball releases;
2) A larger gear on the base axle would allow for finer base angle control;

3) The elbow shaft could be moved out, such that the bouncy ball doesn't impact the upper arm; and

4) The current motors have a plastic gear that had to be pinned through the shaft to prevent slipping; the metal gear on that same shaft may need to be pinned for throwing longer distances.

One area that could increase performance in all aspects would be further work in modeling of the system. A controller is always limited by the accuracy of its governing model. The data on which many of the parameters were based had significant variations; it may prove worthwhile to try using a spectrum analyzer to identify system parameters or to build a mount with which the motor shaft could be clamped, such that the motor inductance could be measured, in the event that it is affecting these results.

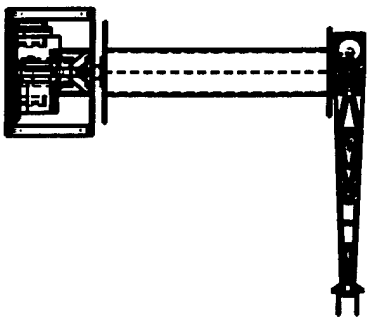
On this same note this controller was designed as something that could be implemented by the PK2600. It has not been optimized for the system. There are more aggressive but computationally demanding techniques, which could be done in dSpace, not intended for use on the PK2600, which could significantly increase the robot's accuracy.

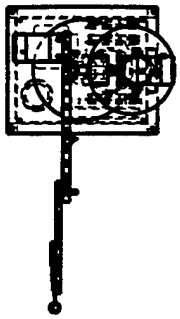
Other things associated with the robot that need to be accomplished include the supporting web page for high school demonstrations. This would include explanations of particle dynamics, air drag, bouncing particles, and more in depth information about the robot.

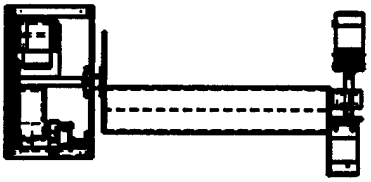
This robot is capable of many exciting demonstrations, it's fast enough that it could potentially juggle balls, get light sensors and hit a light source or any other number of

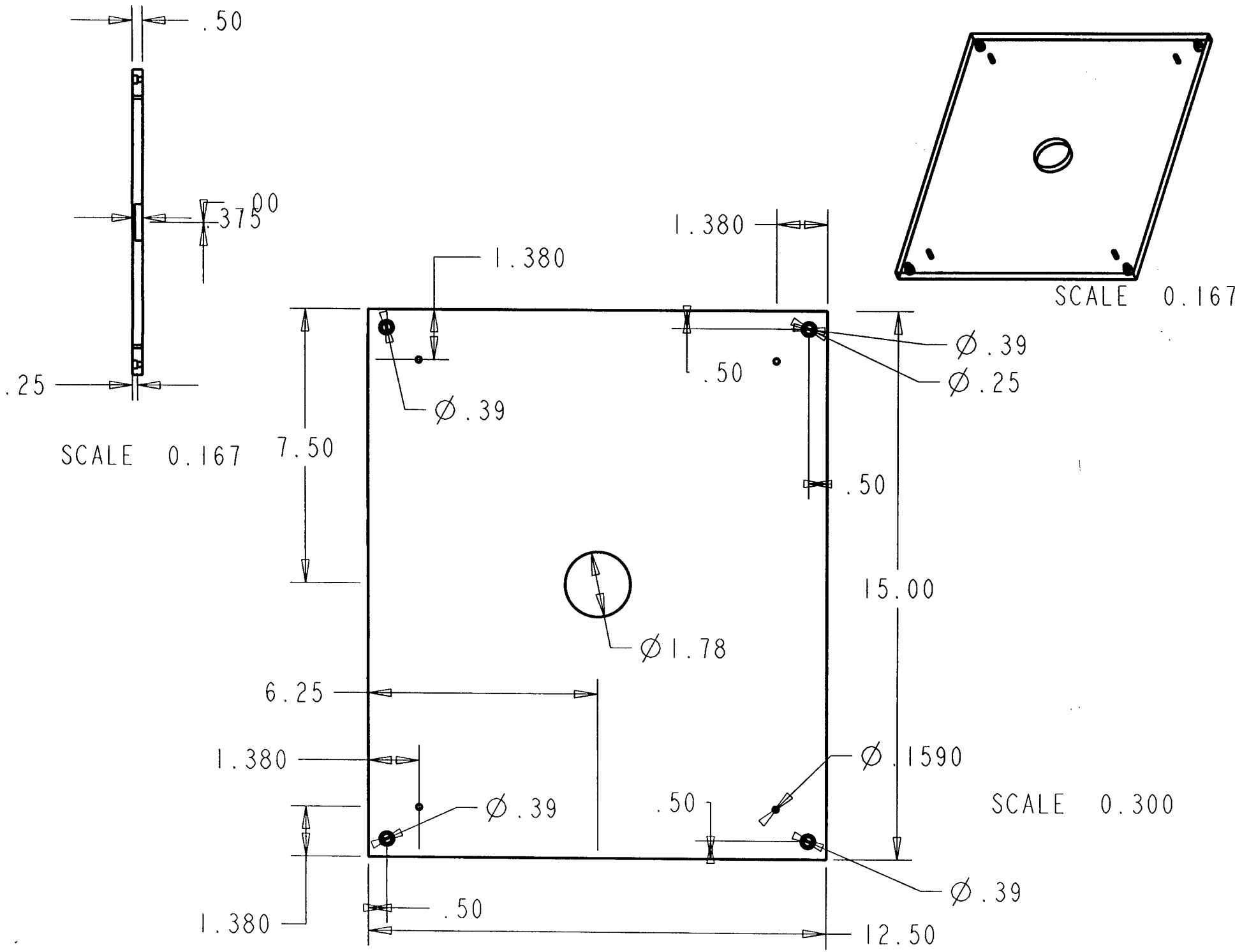
demonstrations that could be imagined. Now that it's built and capable, further programming and use of it remains, which in many ways is the funnest part.

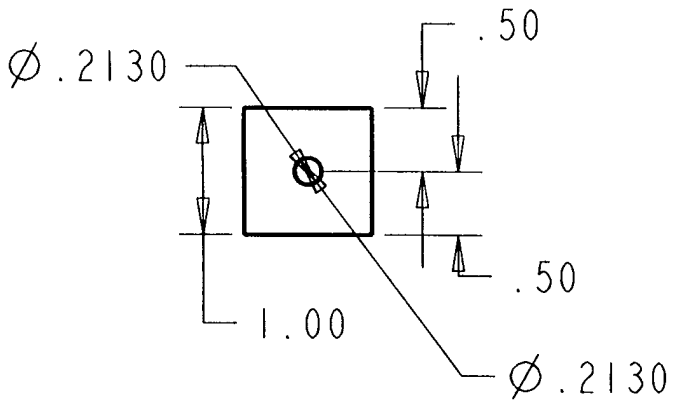
Appendix A Robot Part Sch



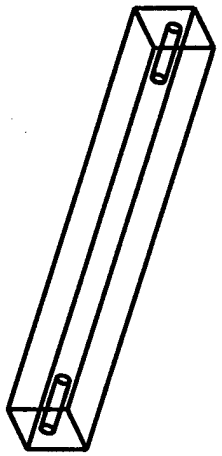




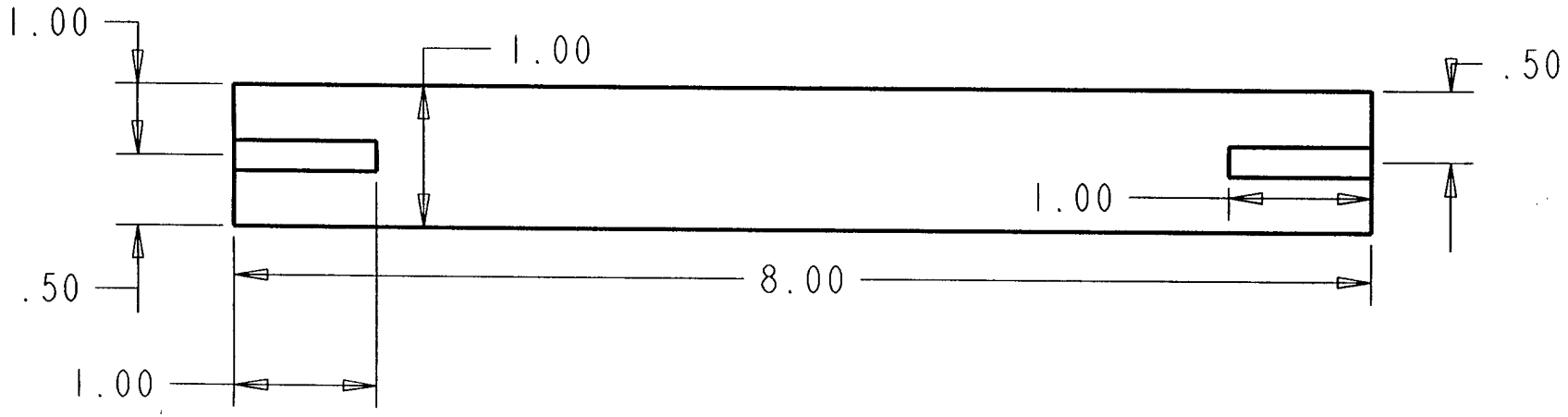

.50
.375
.25
SCALE 0.167
1.380
1.380
.50
Ø.39
Ø.39
Ø.25
7.50
.50
15.00
Ø1.78
6.25
1.380
Ø.39
Ø.1590
.50
SCALE 0.300
Ø.39
1.380
.50
12.50
SCALE 0.167



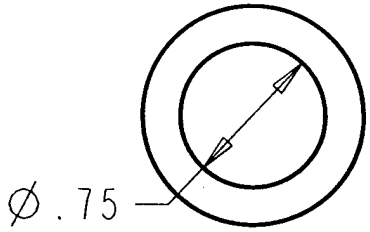
SCALE 0.660



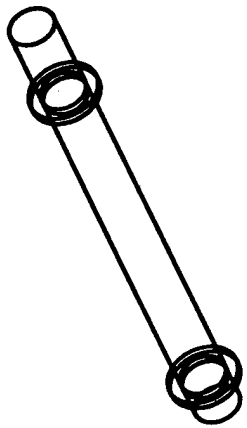
SCALE 0.333



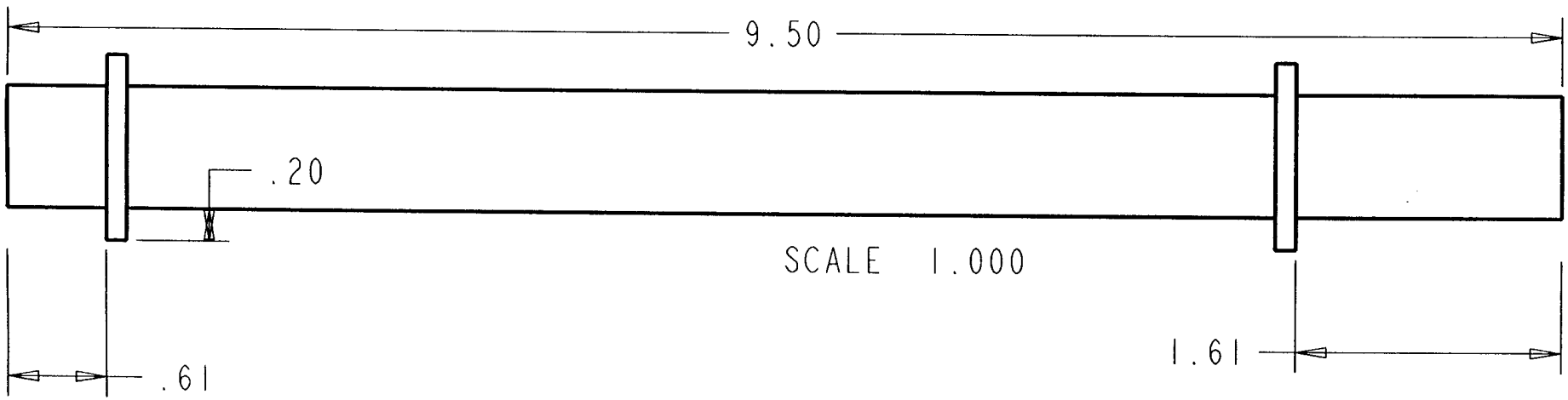
SCALE 0.800



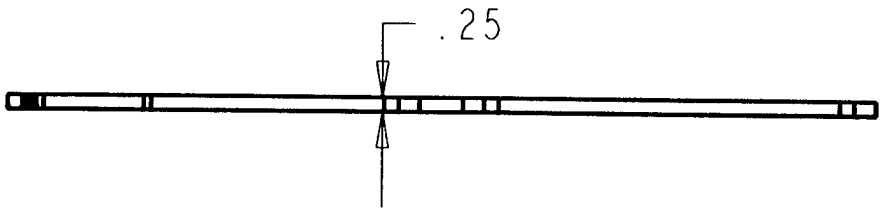
SCALE 1.000



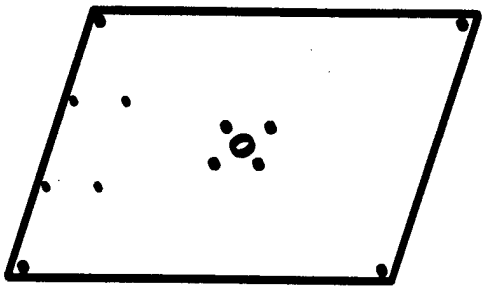
SCALE 0.333



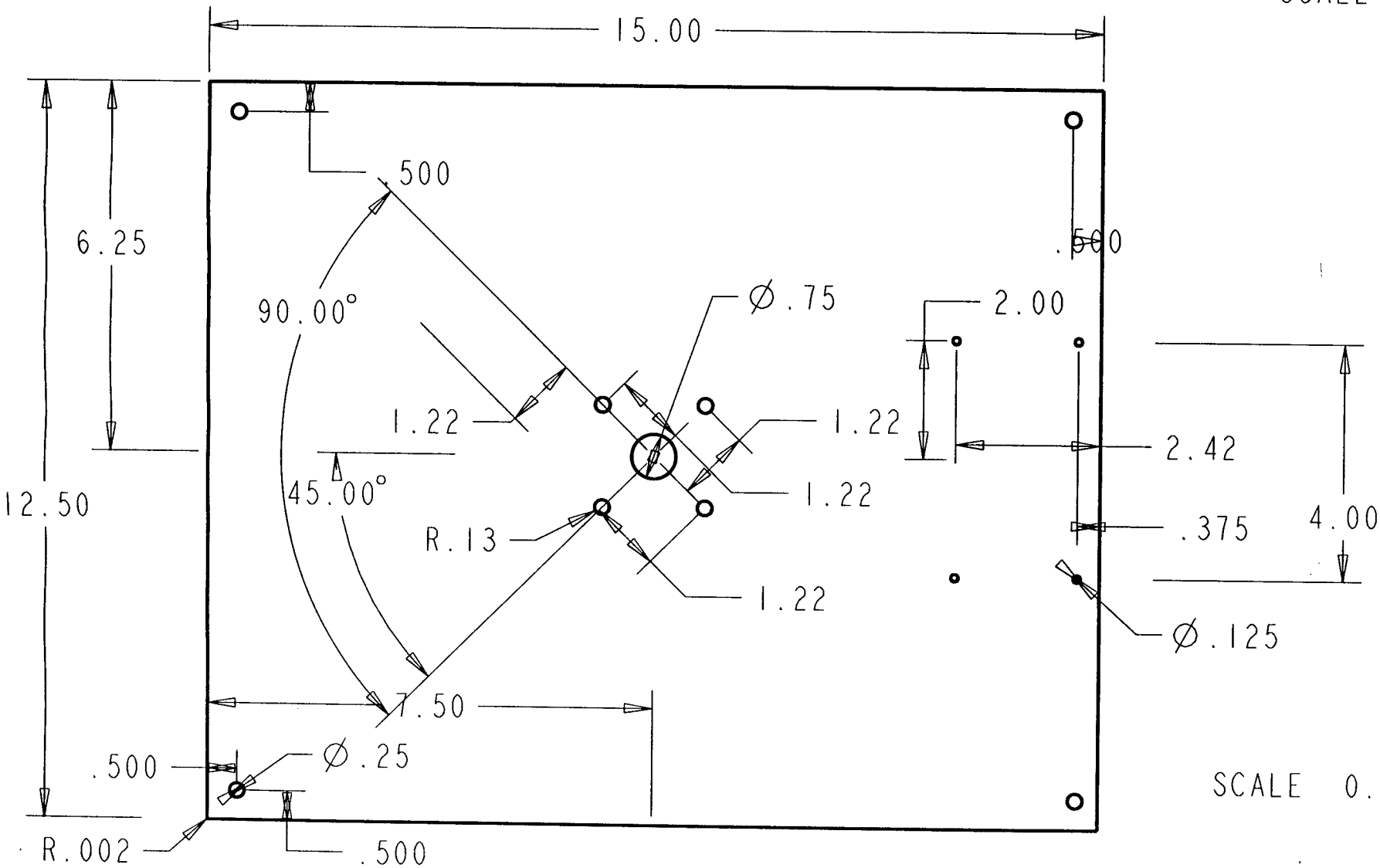
SCALE 1.000



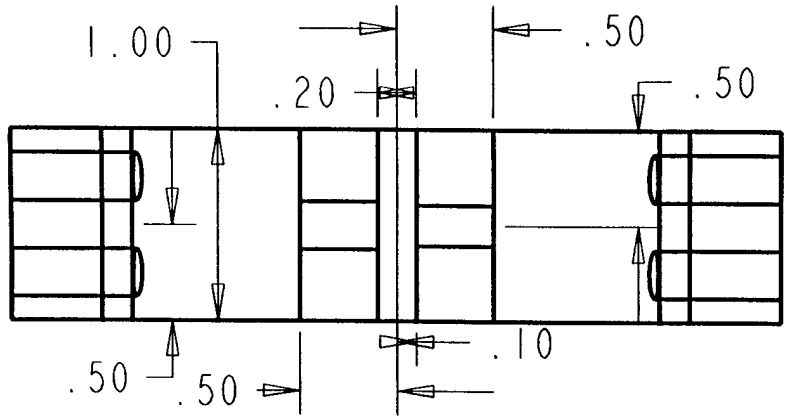
SCALE 0.300



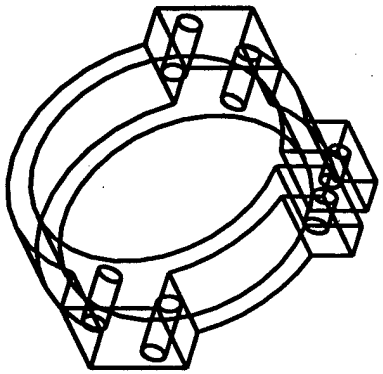
SCALE 0.143



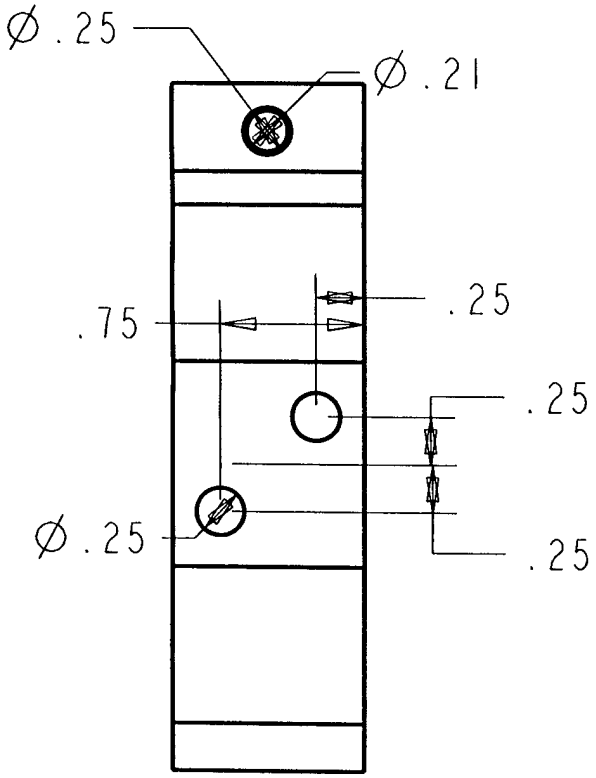
SCALE 0.400



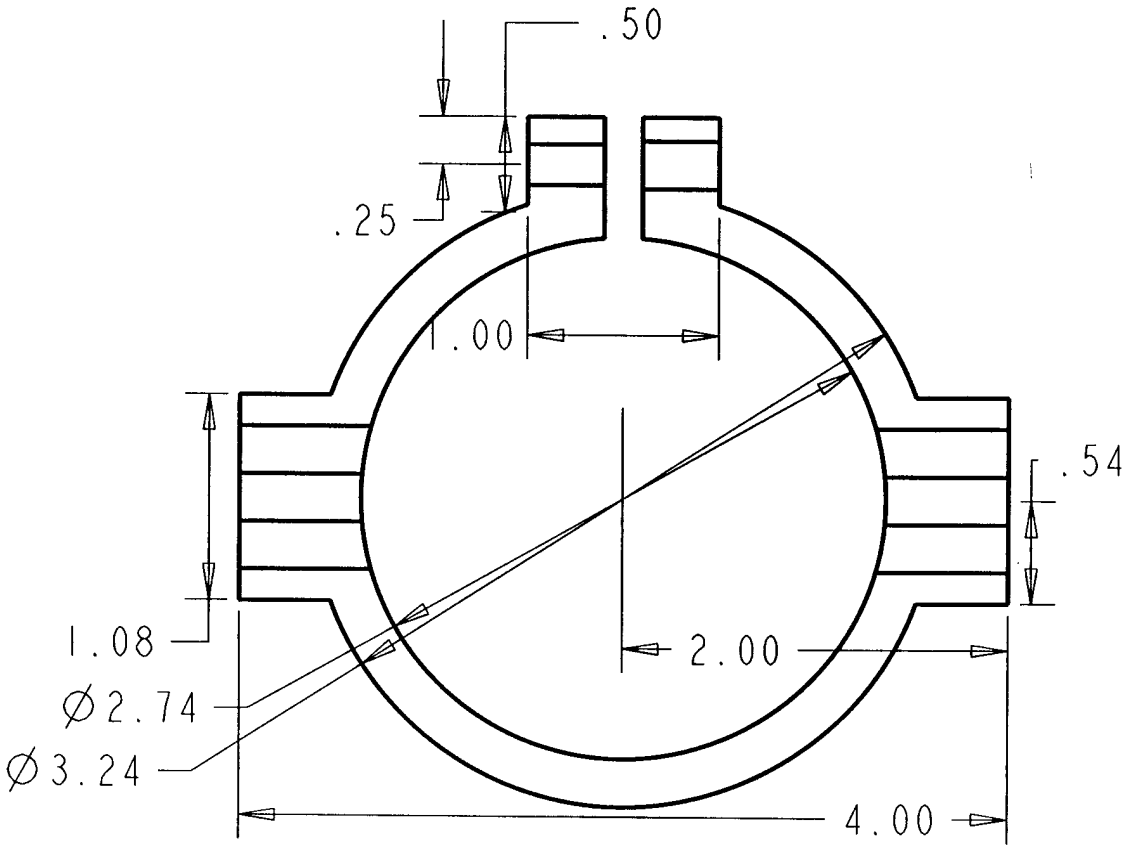
SCALE 1.000



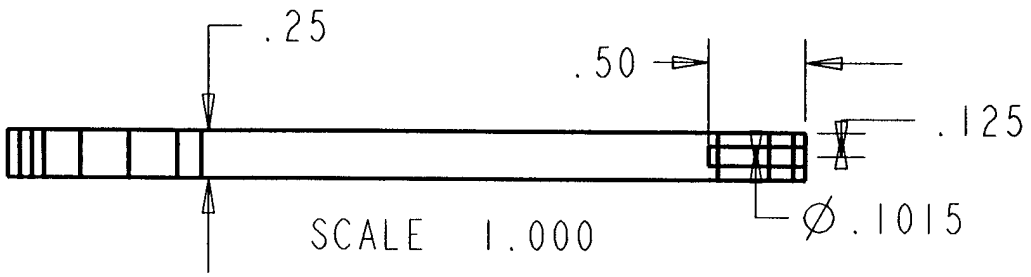
SCALE 0.500



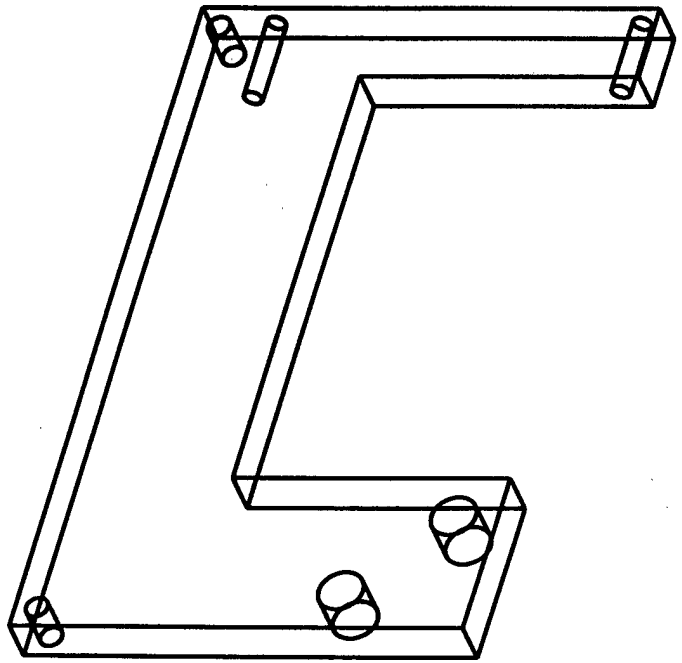
SCALE 1.000



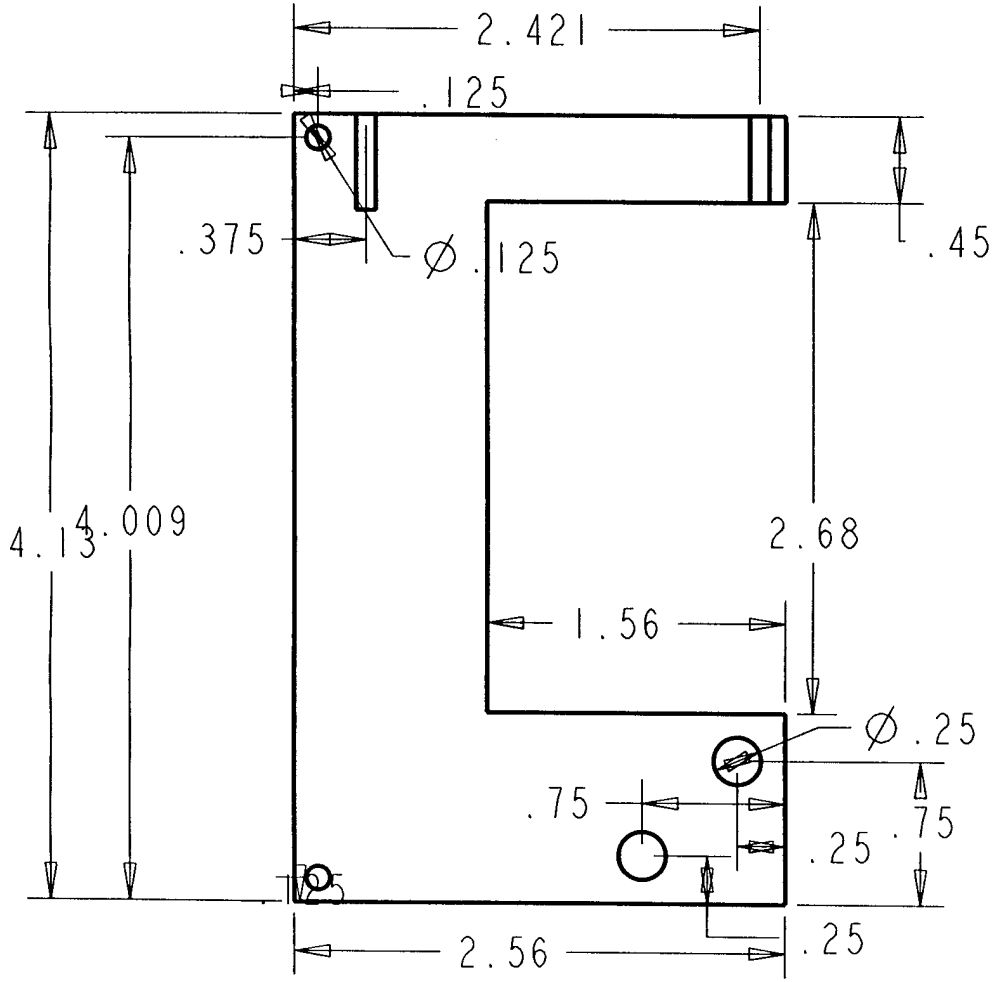
SCALE 1.000



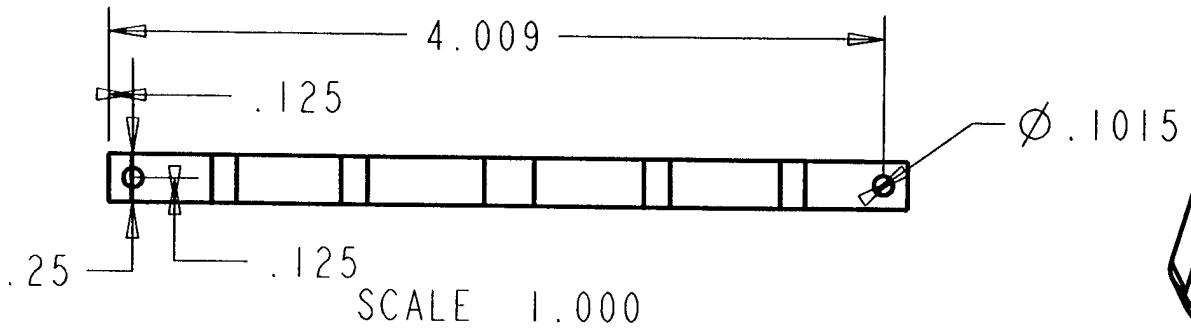
SCALE 1.000



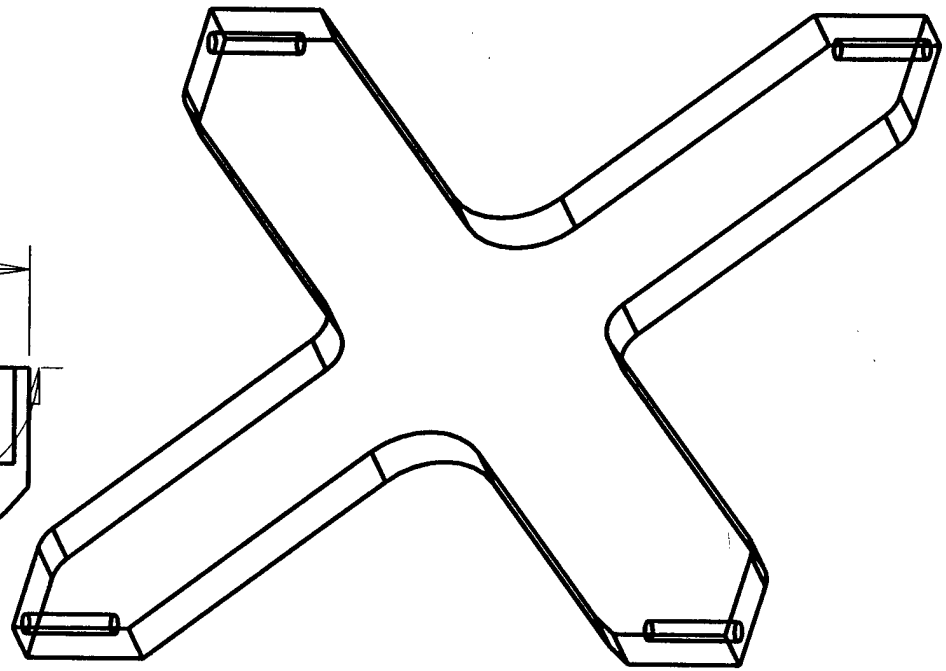
SCALE 1.000



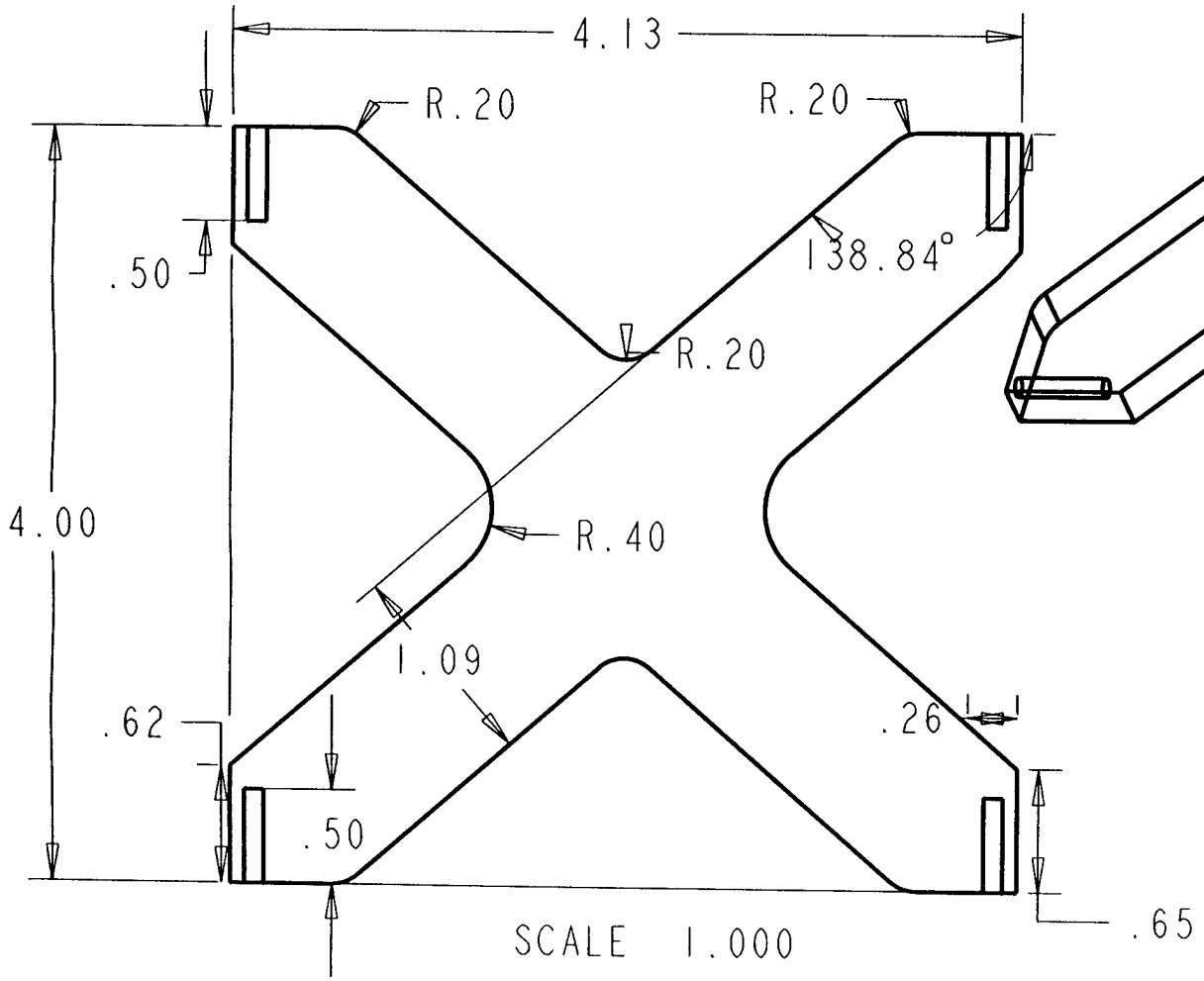
SCALE 1.000



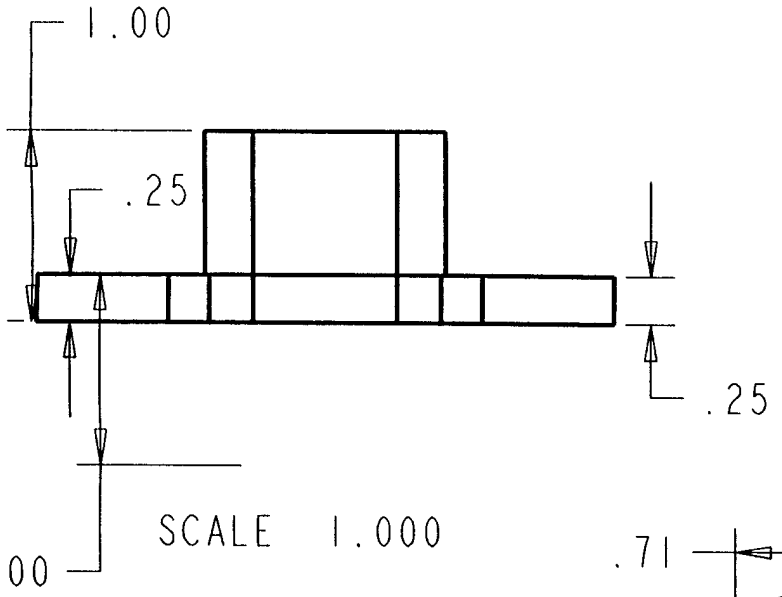
SCALE 1.000



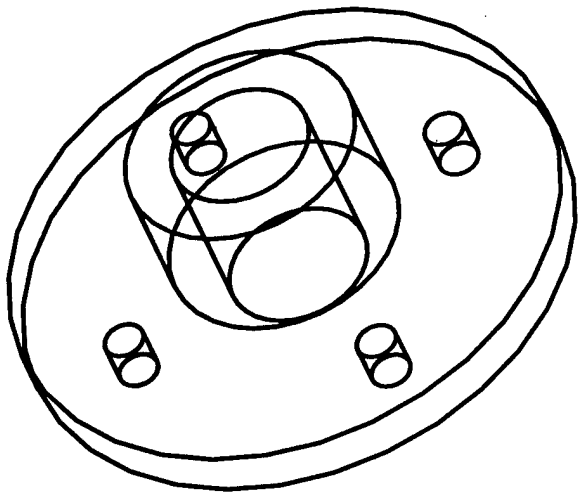
SCALE 1.000



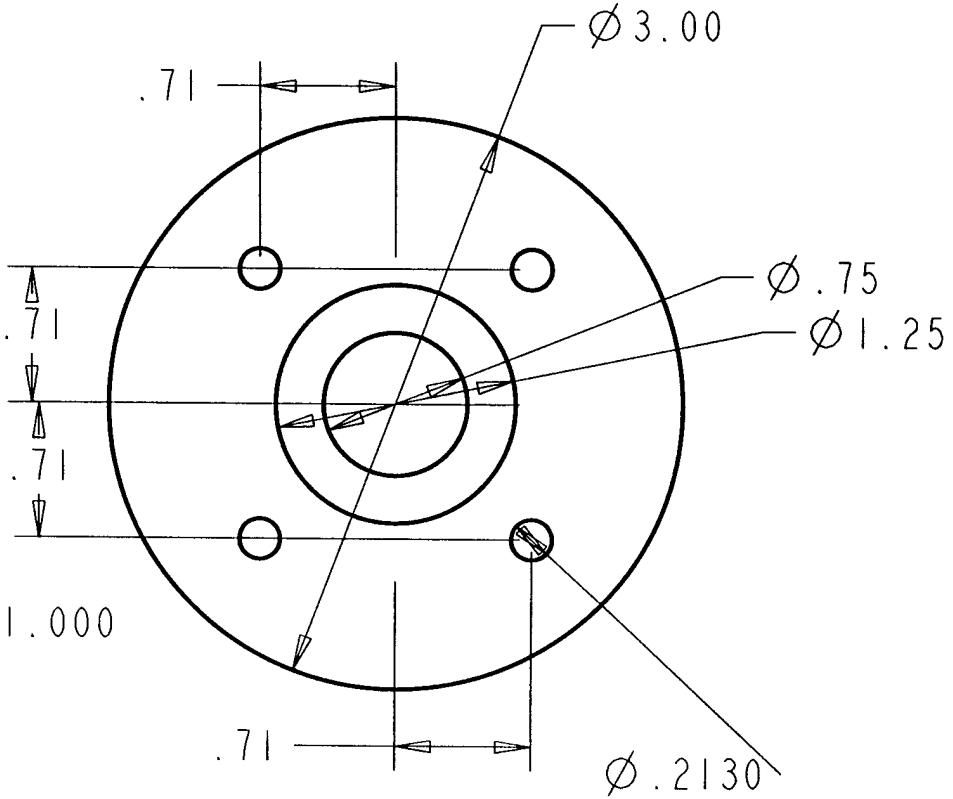
SCALE 1.000



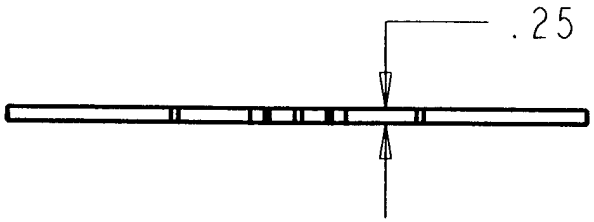
SCALE 1.000



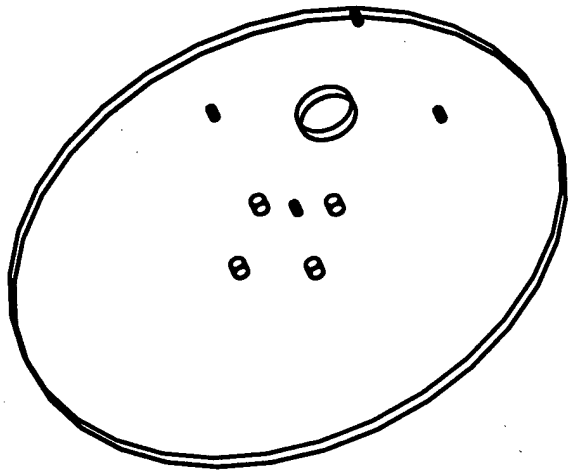
SCALE 1.000



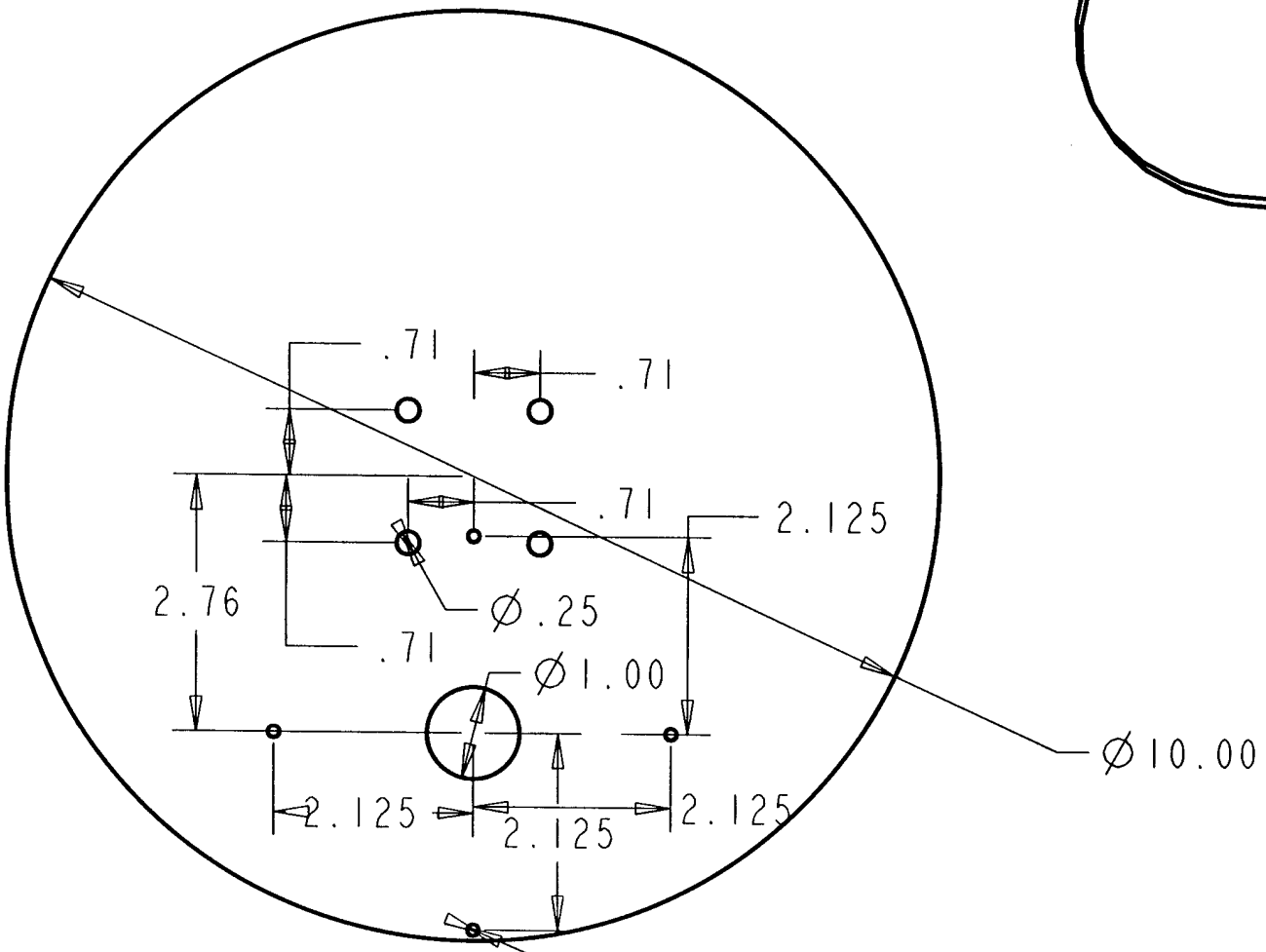
SCALE 1.000



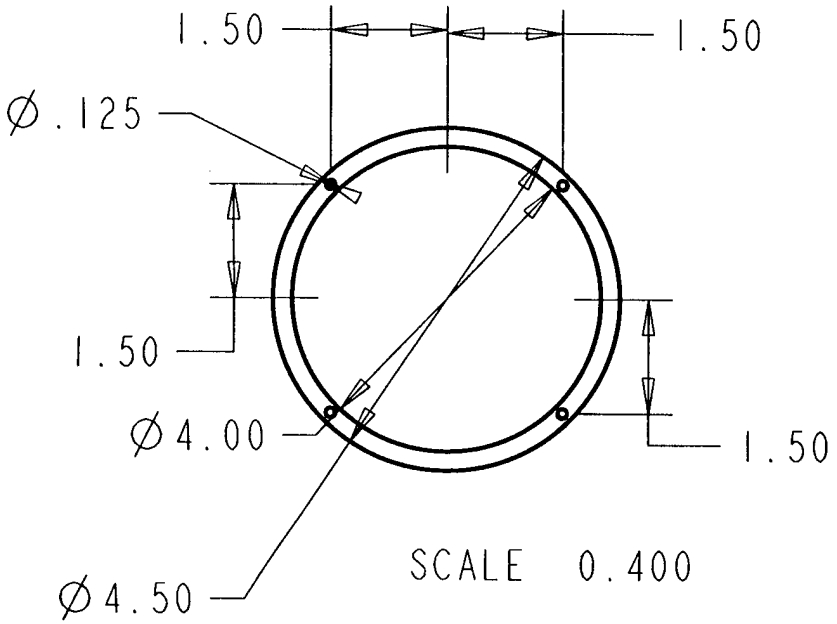
SCALE 0.300



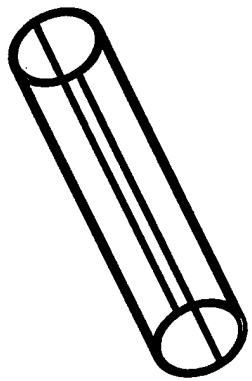
SCALE 0.300



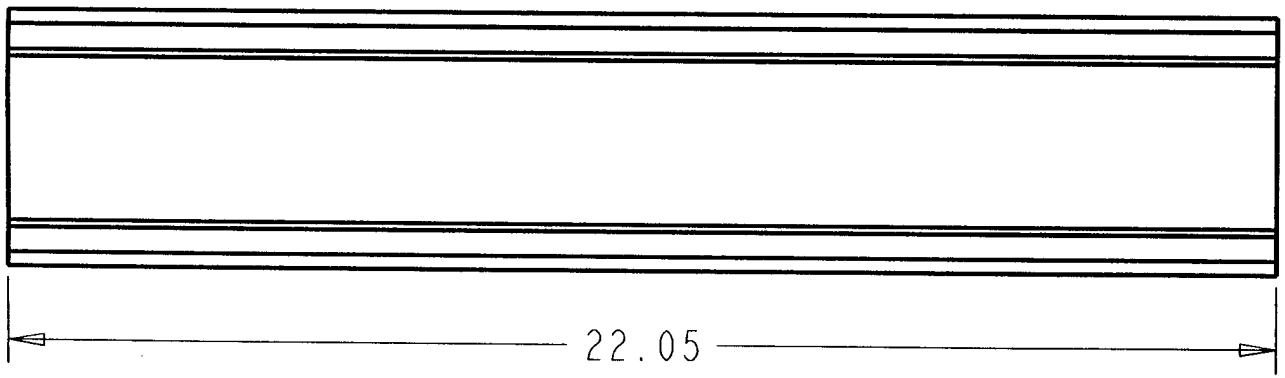
SCALE 0.500



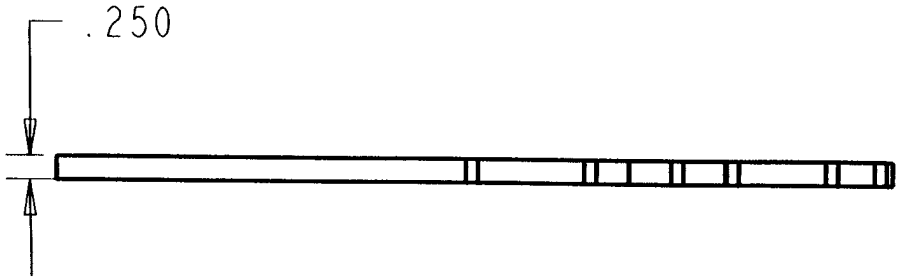
SCALE 0.400



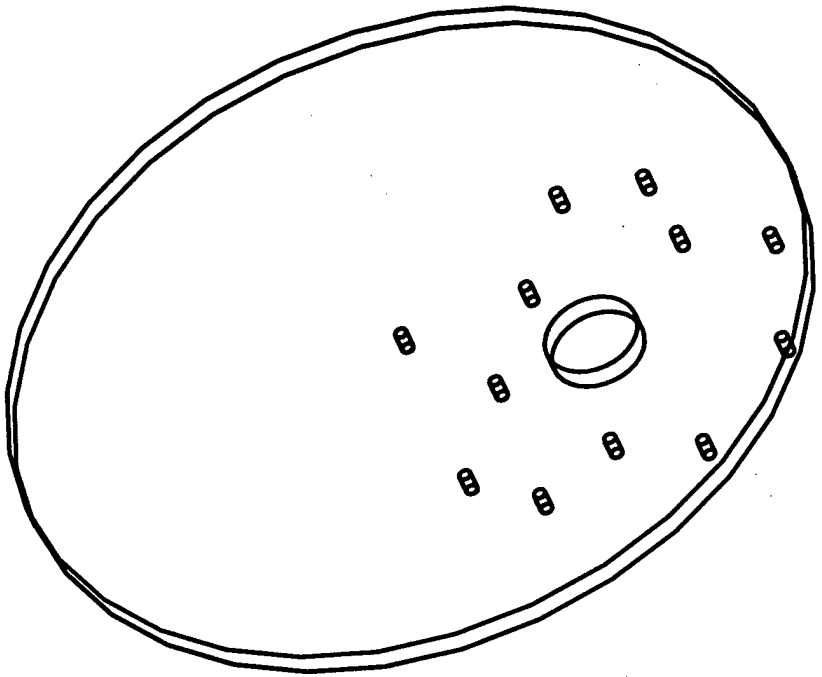
SCALE 0.111



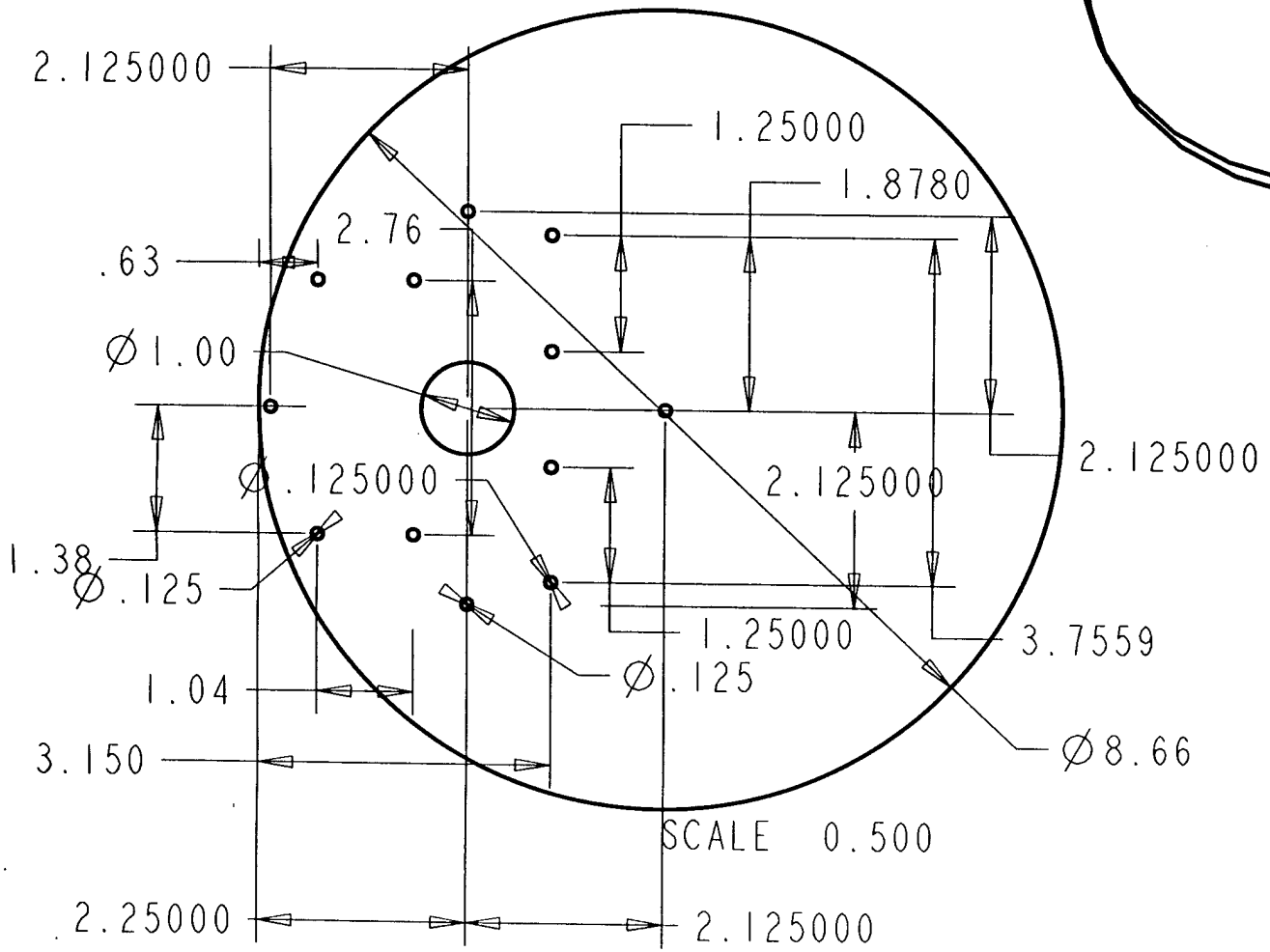
SCALE 0.300



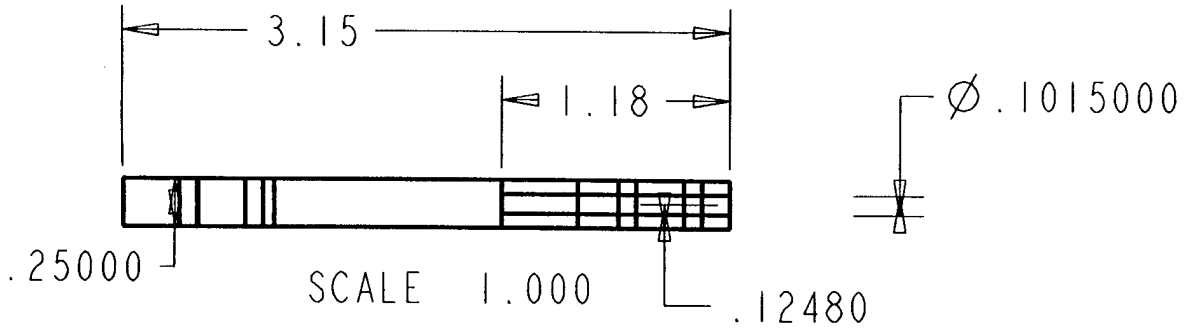
SCALE 0.500

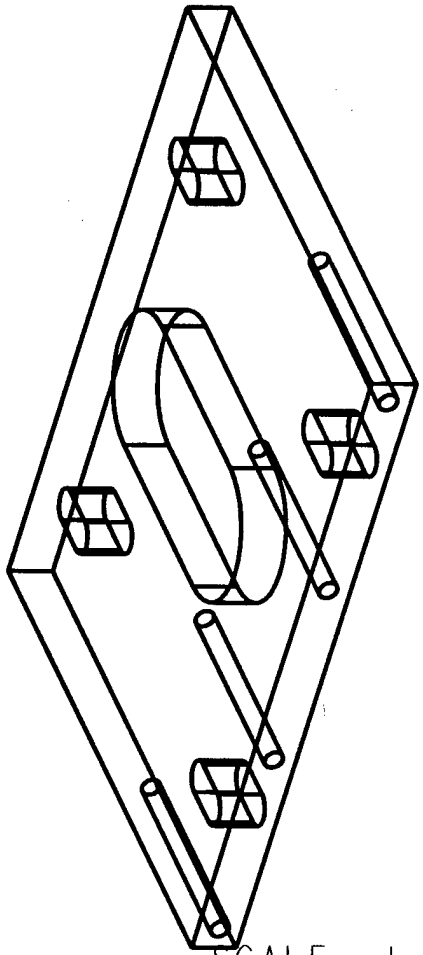


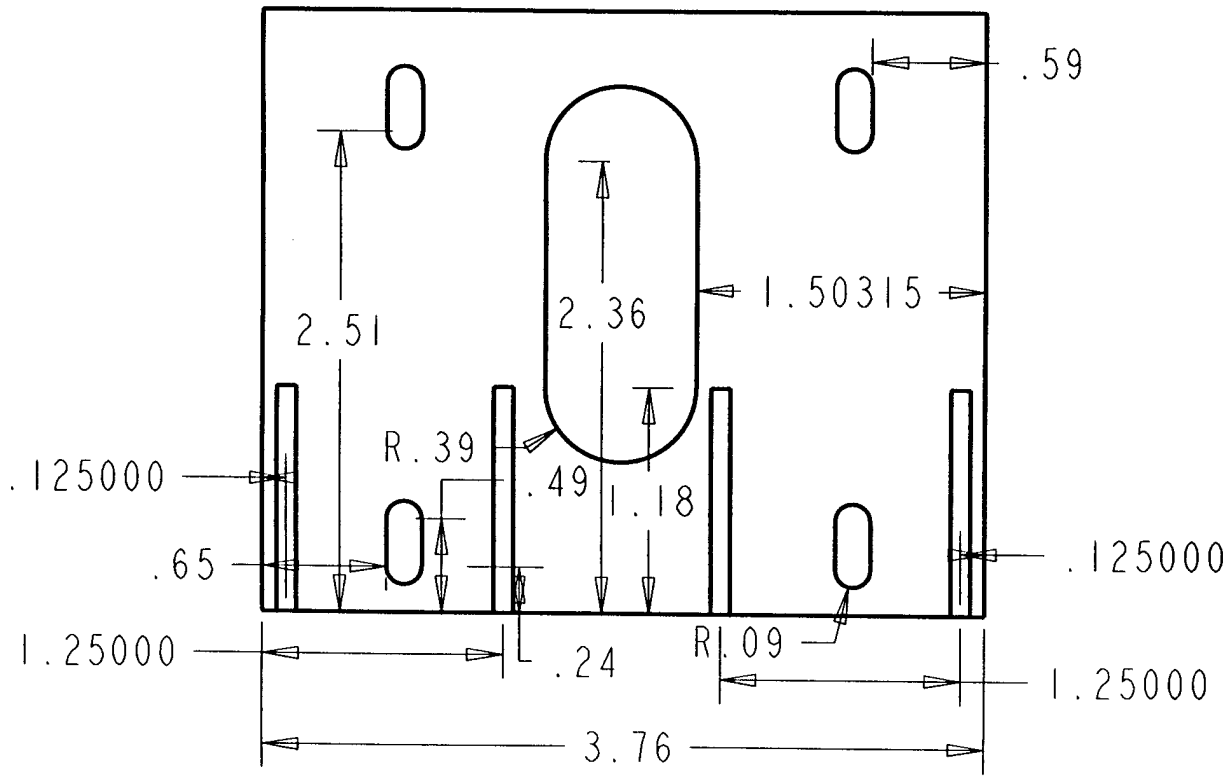
SCALE 0.500

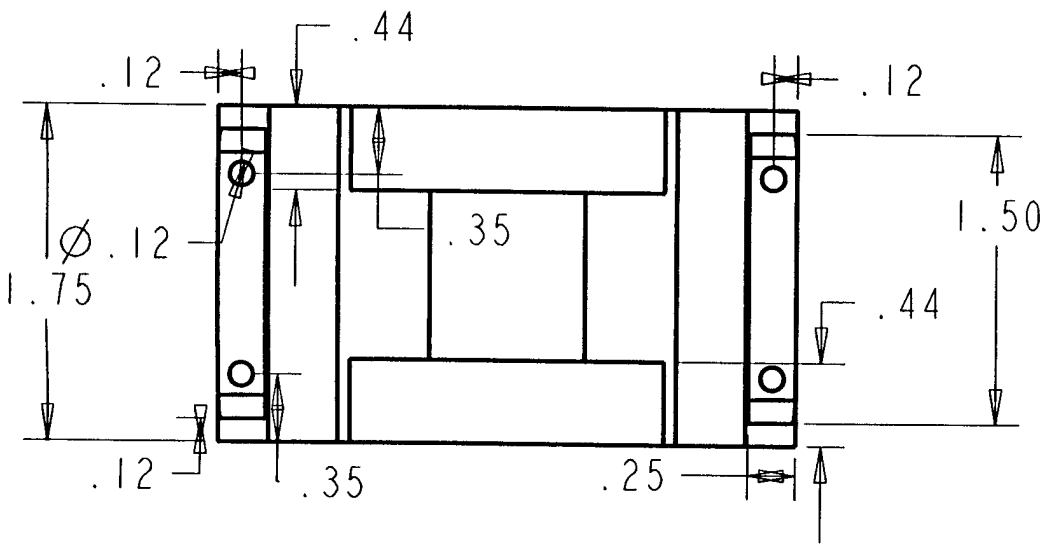


SCALE 0.500

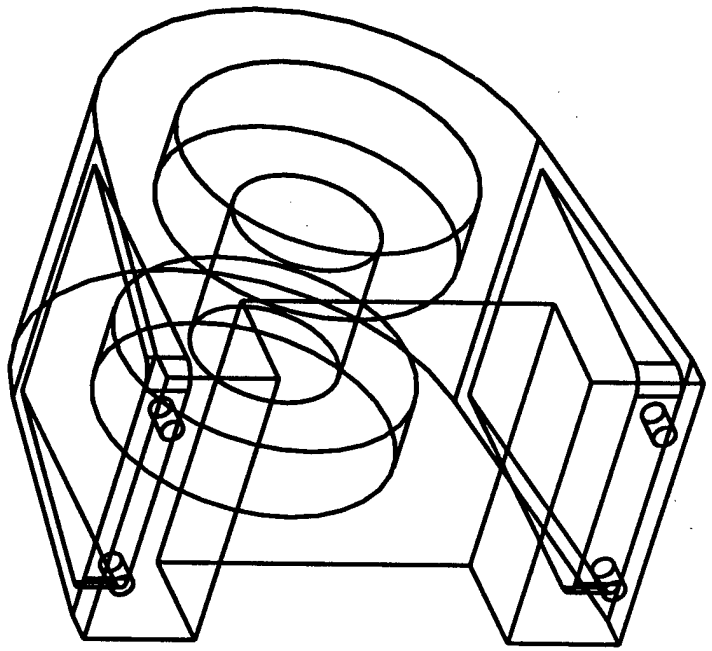

3.15
1.18
Ø.1015000
.25000
SCALE 1.000
.12480


SCALE 1.000

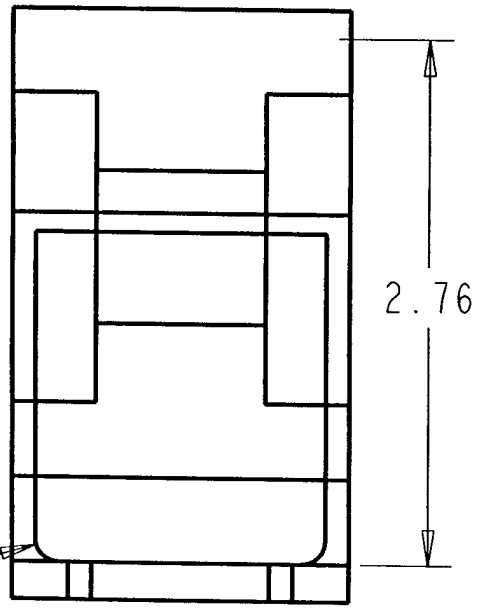

.59
2.51
2.36
1.50315
R.39
.49
1.18
.125000
.65
.125000
1.25000
.24
R.09
1.25000
3.76
SCALE 1.000



SCALE 1.000

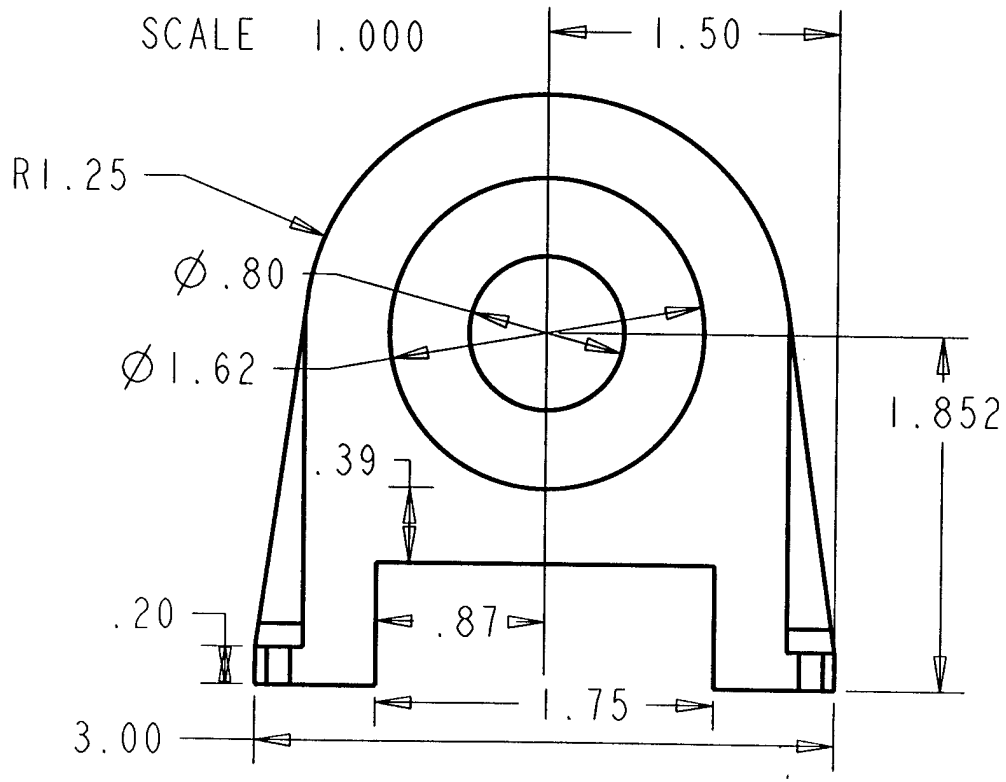


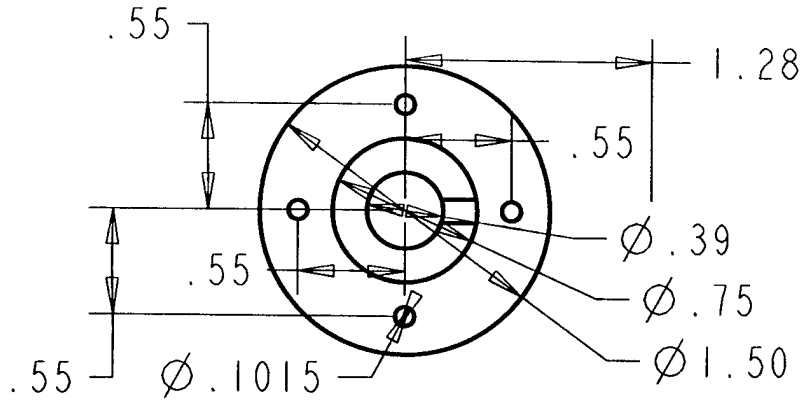
SCALE 1.000



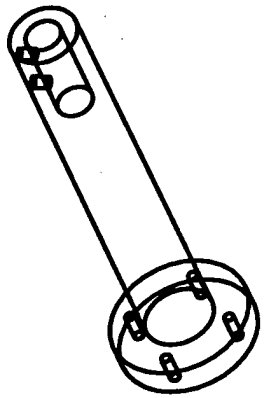
SCALE 1.000

SCALE 1.000

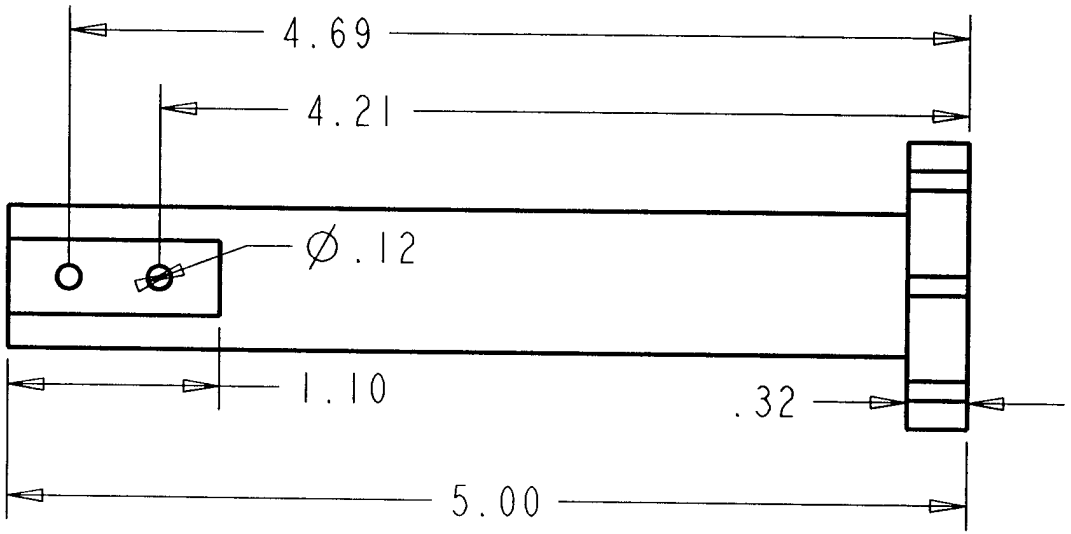




SCALE 1.000



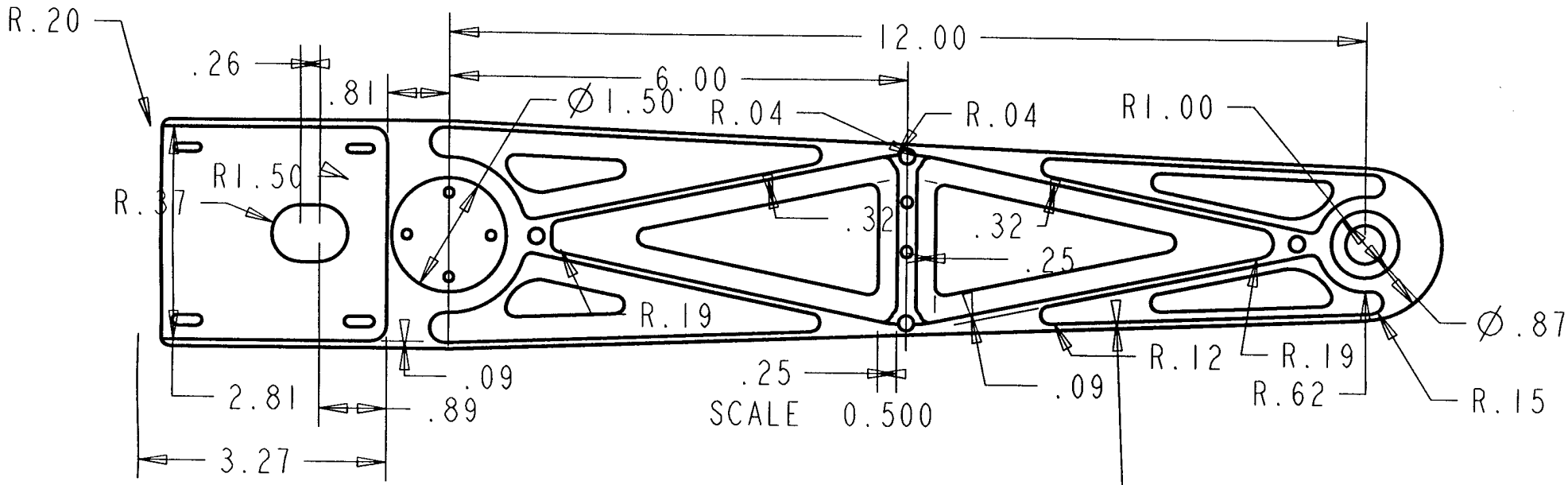
SCALE 0.500

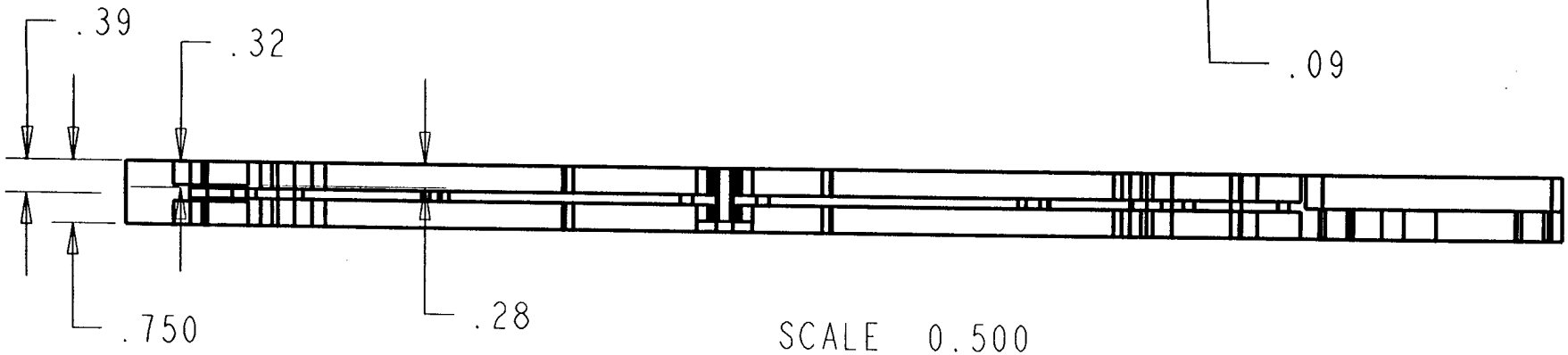


SCALE 1.000

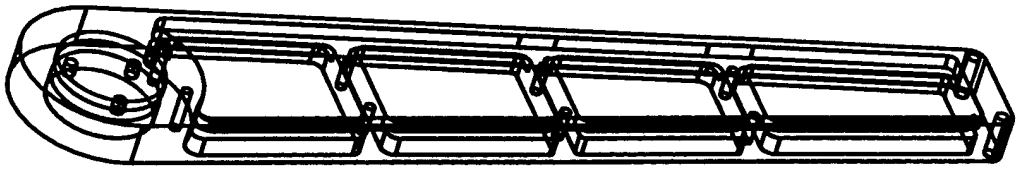
SCALE 0.500

SCALE 0.300

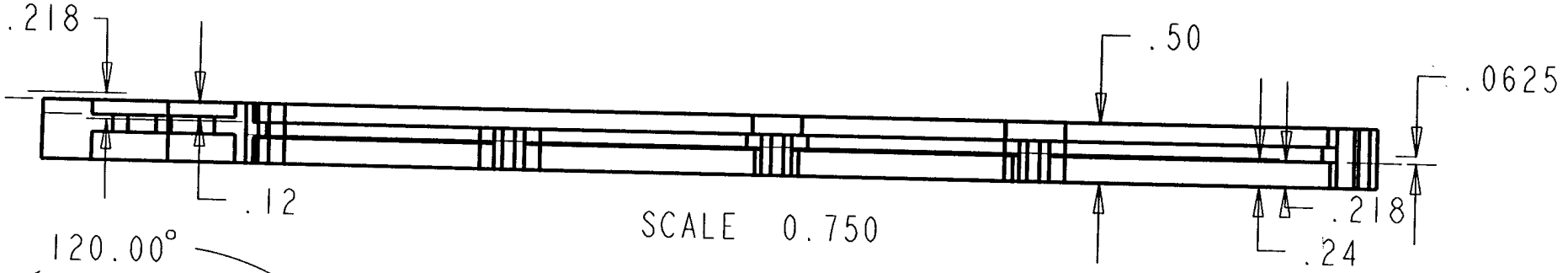




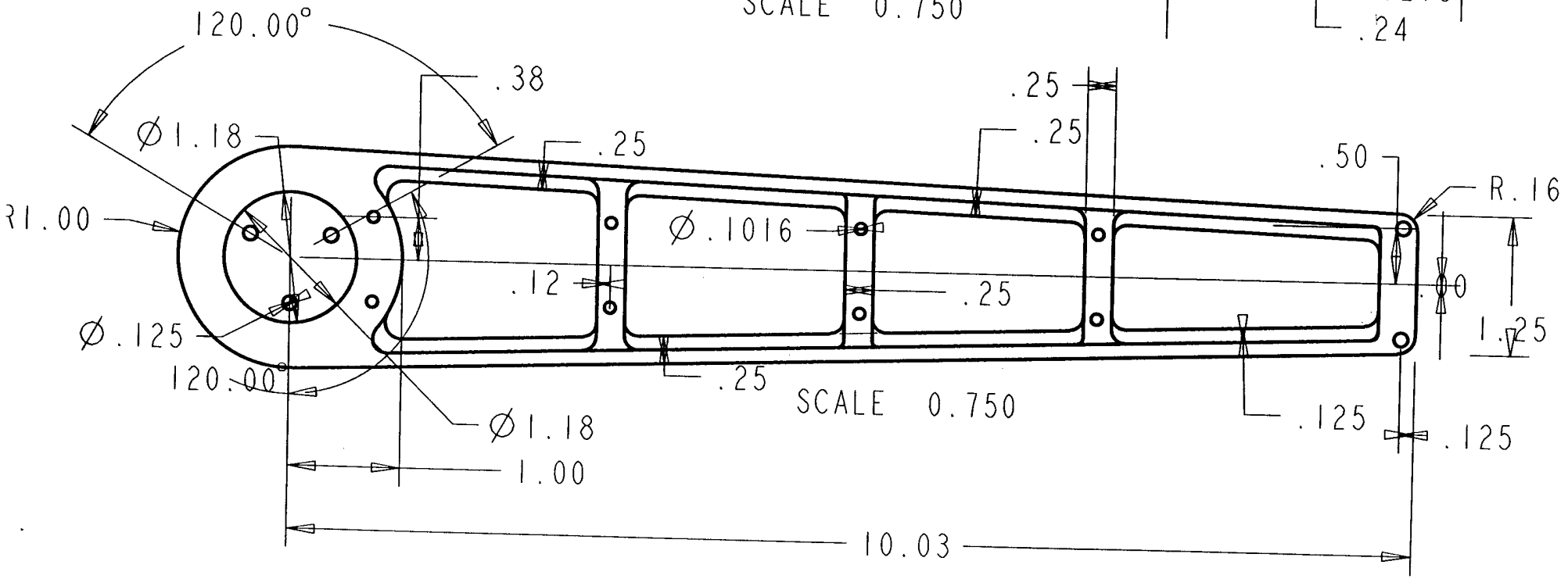
SCALE 0.500



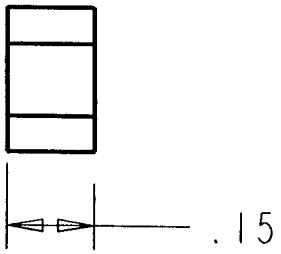
SCALE 0.500



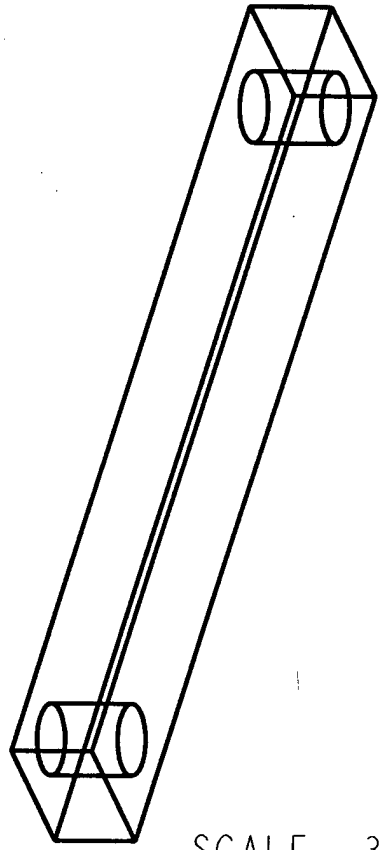
SCALE 0.750



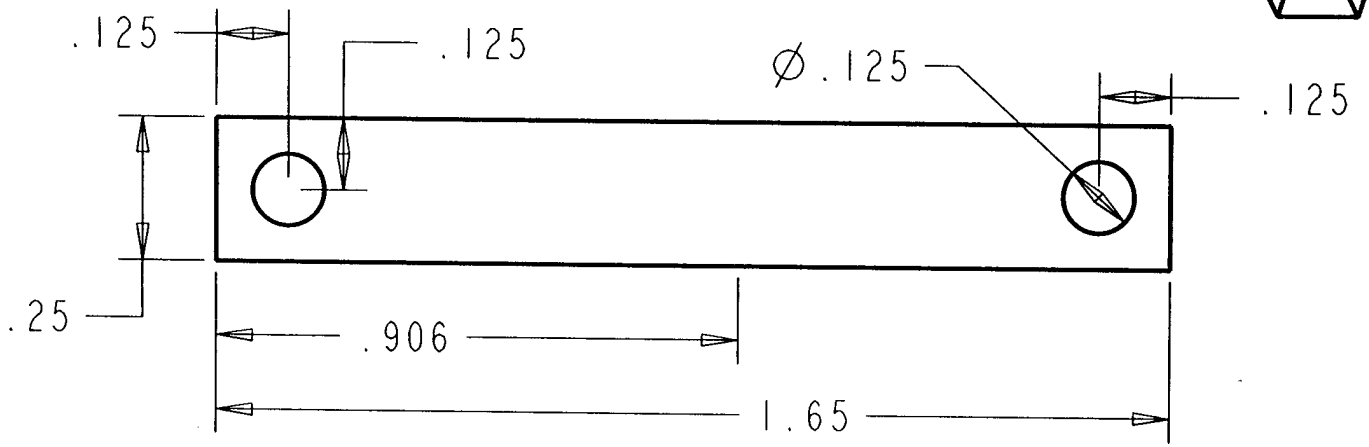
SCALE 0.750



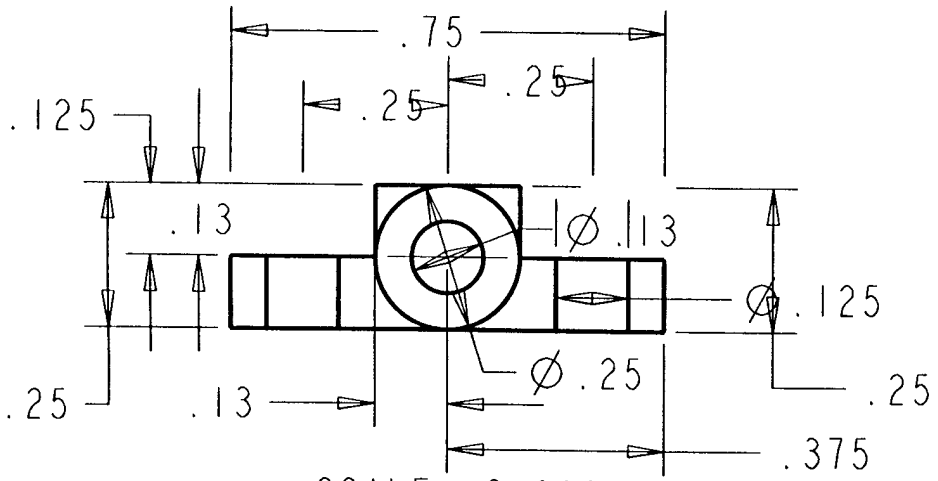
SCALE 3.000



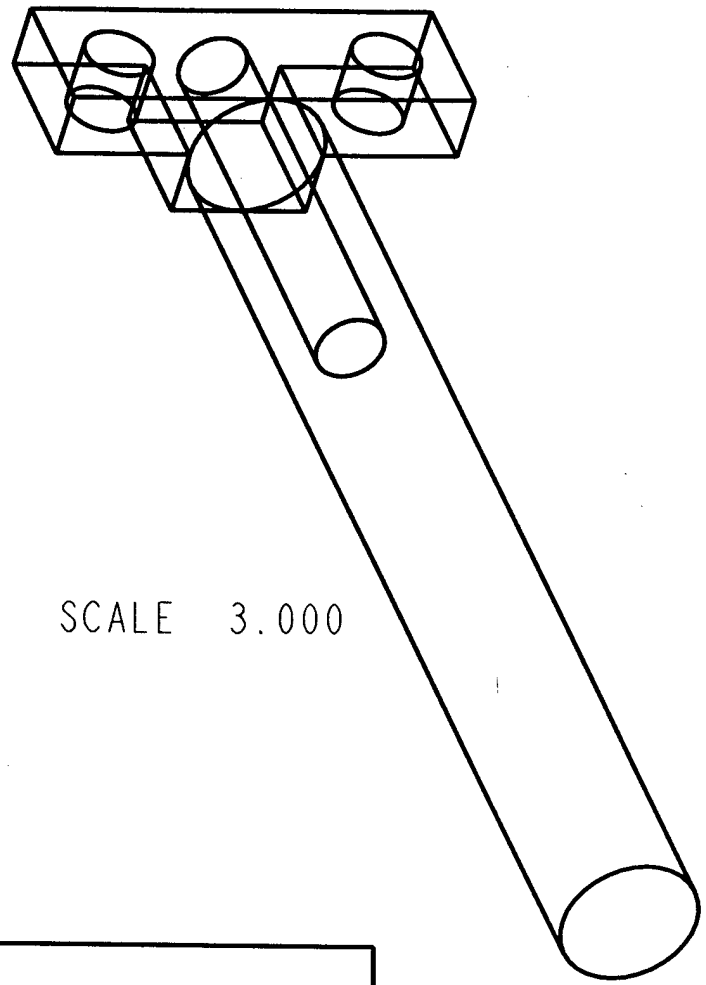
SCALE 3.000



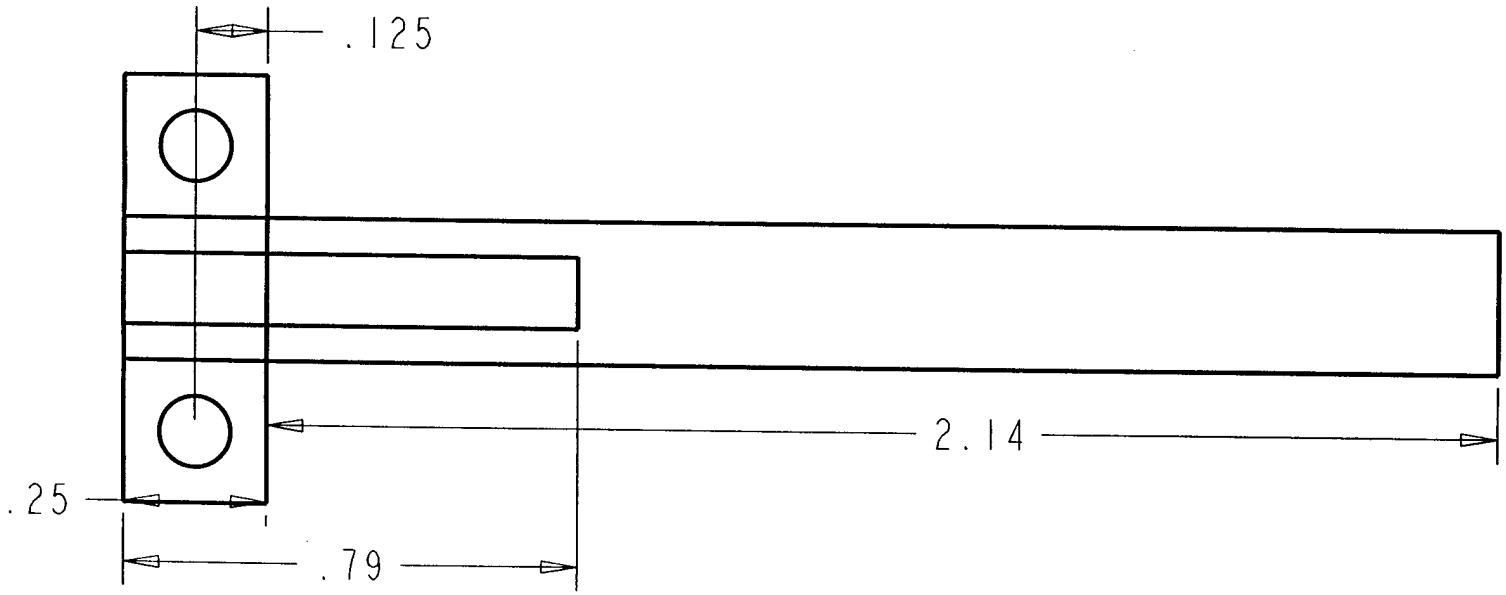
SCALE 3.000



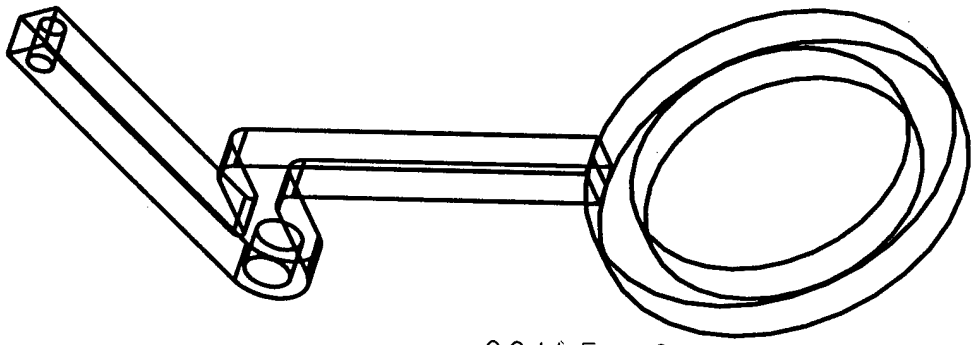
SCALE 3.000



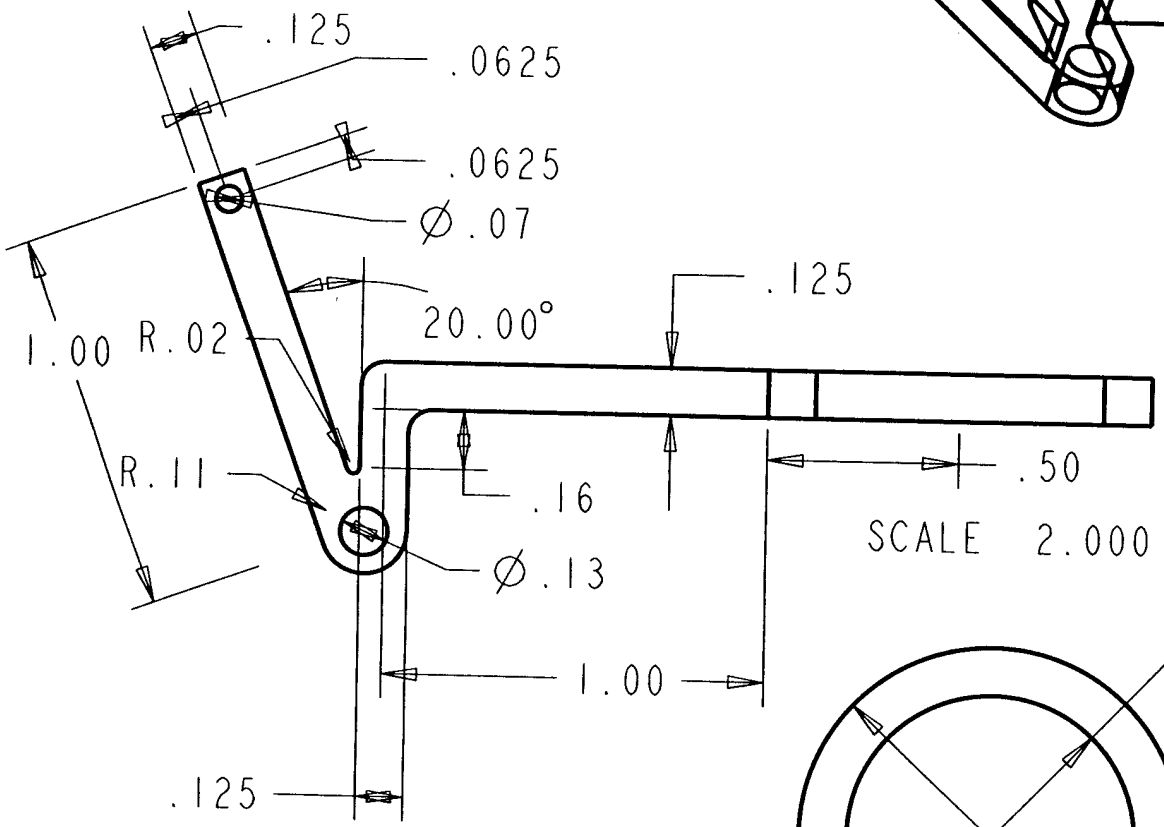
SCALE 3.000



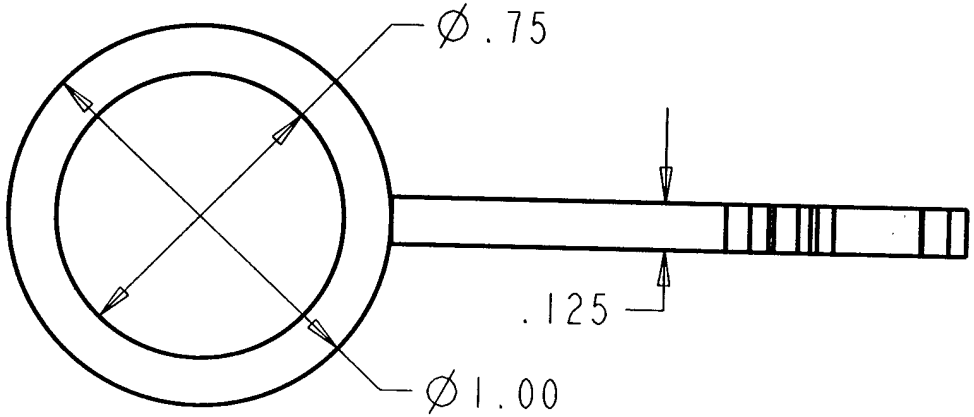
SCALE 3.000



SCALE 2.000



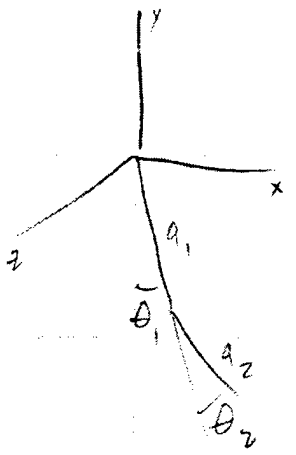
SCALE 2.000



SCALE 2.000

Appendix B Kinematic and Torque Equation Derivations

The kinematic and torque equations were first derived by hand and then checked using Maple. As inertial parameters were updated Maple was used for quick recalculation. Original hand calc's and verifying Maple output are included. Free body diagram included with hand calculations.



$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 s_1 + a_2 s_{12} \\ -a_1 c_1 - a_2 c_{12} \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} a_1 c_1 \dot{\theta}_1 + a_2 c_{12} \dot{\theta}_1 + a_2 c_{12} \dot{\theta}_2 \\ a_1 s_1 \dot{\theta}_1 + a_2 s_{12} \dot{\theta}_1 + a_2 s_{12} \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ a_1 s_1 + a_2 s_{12} & a_2 s_{12} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \frac{a_{22}}{a_{11}a_{22} - a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22} - a_{12}a_{21}} \\ \frac{-a_{21}}{a_{11}a_{22} - a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22} - a_{12}a_{21}} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

```
>Position:=linalg[matrix](2,1,[a1*sin(theta1(t))+a2*sin(theta1(t)+theta2
(t)),-a1*cos(theta1(t))-a2*cos(theta1(t)+theta2(t))]);
```

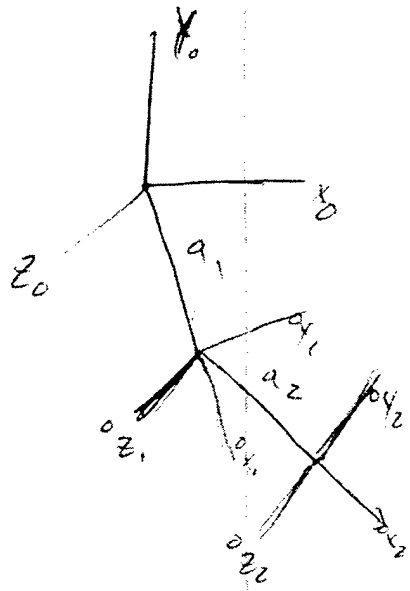
$$Position := \begin{bmatrix} a1\sin(\theta1(t)) + a2\sin(\theta1(t)+\theta2(t)) \\ -a1\cos(\theta1(t)) - a2\cos(\theta1(t)+\theta2(t)) \end{bmatrix}$$

```
>Velocity:=map(diff,Position,t);
```

$$Velocity := \begin{bmatrix} a1\cos(\theta1(t))\left(\frac{\partial}{\partial t}\theta1(t)\right) + a2\cos(\theta1(t)+\theta2(t))\left(\left(\frac{\partial}{\partial t}\theta1(t)\right)+\left(\frac{\partial}{\partial t}\theta2(t)\right)\right) \\ a1\sin(\theta1(t))\left(\frac{\partial}{\partial t}\theta1(t)\right) + a2\sin(\theta1(t)+\theta2(t))\left(\left(\frac{\partial}{\partial t}\theta1(t)\right)+\left(\frac{\partial}{\partial t}\theta2(t)\right)\right) \end{bmatrix}$$

```
>Acceleration:=map(diff,Velocity,t);
```

$$Acceleration := \begin{bmatrix} -a1\sin(\theta1(t))\left(\frac{\partial}{\partial t}\theta1(t)\right)^2 + a1\cos(\theta1(t))\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right) \\ -a2\sin(\theta1(t)+\theta2(t))\left(\left(\frac{\partial}{\partial t}\theta1(t)\right)+\left(\frac{\partial}{\partial t}\theta2(t)\right)\right)^2 \\ +a2\cos(\theta1(t)+\theta2(t))\left(\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)\right) \\ a1\cos(\theta1(t))\left(\frac{\partial}{\partial t}\theta1(t)\right)^2 + a1\sin(\theta1(t))\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right) \\ +a2\cos(\theta1(t)+\theta2(t))\left(\left(\frac{\partial}{\partial t}\theta1(t)\right)+\left(\frac{\partial}{\partial t}\theta2(t)\right)\right)^2 \\ +a2\sin(\theta1(t)+\theta2(t))\left(\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)\right) \end{bmatrix}$$



i	a_i	d_i	α_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2

$${}^0\dot{\omega}_1 = \ddot{\theta}_1\,{}^0z_0$$

$${}^0\omega_2 = {}^0\dot{\omega}_1 + \ddot{\theta}_2\,{}^0z_1 + \dot{\theta}_2\,{}^0\omega_1 \times {}^0z_1$$

$$= {}^0\dot{\omega}_1 + \ddot{\theta}_2\,{}^0z_1 + \dot{\theta}_2\dot{\theta}_1\,{}^0z_0 \times {}^0z_1 \quad \Phi$$

$$= \ddot{\theta}_1\,{}^0z_0 + \ddot{\theta}_2\,{}^0z_1$$

$$= \ddot{\theta}_1\,{}^0z_0 + \ddot{\theta}_2\,{}^0z_0$$

$${}^0v_1 = {}^0\dot{\omega}_1 \times {}^0d_{01} + {}^0\omega_1 \times ({}^0\omega_1 \times {}^0d_{01})$$

$${}^0v_2 = {}^0\dot{\omega}_2 \times {}^0d_{12} + {}^0\omega_2 \times ({}^0\omega_2 \times {}^0d_{12}) + {}^0v_1$$

~~${}^0v_1 = \ddot{\theta}_1\,{}^0z_0 \times a_1\,{}^0x_1 + (\dot{\theta}_1+\dot{\theta}_2)\,{}^0z_0 \times (\dot{\theta}_1+\dot{\theta}_2)$~~

$${}^0v_1 = \ddot{\theta}_1\,{}^0z_0 \times a_1\,{}^0x_1 + \dot{\theta}_1\,{}^0z_0 \times \dot{\theta}_1\,{}^0z_0 \times a_1\,{}^0x_1$$

$$= \ddot{\theta}_1 a_1\,{}^0y_1 + \dot{\theta}_1\,{}^0z_0 \times \dot{\theta}_1 a_1\,{}^0y_1$$

$$= \ddot{\theta}_1 a_1\,{}^0y_1 - \dot{\theta}_1^2 a_1\,{}^0x_1$$

$${}^0v_2 = (\ddot{\theta}_1\,{}^0z_0 + \ddot{\theta}_2\,{}^0z_0) \times a_2\,{}^0x_2 + (\dot{\theta}_1\,{}^0z_0 + \dot{\theta}_2\,{}^0z_0) \times (\dot{\theta}_1\,{}^0z_0 + \dot{\theta}_2\,{}^0z_0) \times a_2\,{}^0x_2 + \ddot{\theta}_1 a_1\,{}^0y_1 - \dot{\theta}_1^2 a_1\,{}^0x_1$$

$$= (\ddot{\theta}_1+\ddot{\theta}_2)a_2\,{}^0y_2 + (\dot{\theta}_1\,{}^0z_0 + \dot{\theta}_2\,{}^0z_0) \times (\dot{\theta}_1+\dot{\theta}_2)a_2\,{}^0y_2 + \ddot{\theta}_1 a_1\,{}^0y_1 - \dot{\theta}_1^2 a_1\,{}^0x_1$$

$$= (\ddot{\theta}_1+\ddot{\theta}_2)a_2\,{}^0y_2 - (\dot{\theta}_1+\dot{\theta}_2)^2 a_2\,{}^0x_2 + \ddot{\theta}_1 a_1\,{}^0y_1 - \dot{\theta}_1^2 a_1\,{}^0x_1$$

$$\ddot{x} = -(\ddot{\theta}_1+\ddot{\theta}_2)a_2 c_{12} - (\dot{\theta}_1+\dot{\theta}_2)^2 a_2 s_{12} + \ddot{\theta}_1 a_1 c_1 - \dot{\theta}_1^2 a_1 s_1$$

$$= -(\ddot{\theta}_1+\ddot{\theta}_2)a_2 s_{12} - \dot{\theta}_1^2 a_1 s_1 + \begin{bmatrix} a_2 c_{12} + a_1 c_1 & a_2 c_{12} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

$$\ddot{y} = (\ddot{\theta}_1+\ddot{\theta}_2)a_2 s_{12} + (\dot{\theta}_1+\dot{\theta}_2)^2 a_2 c_{12} + \ddot{\theta}_1 a_1 s_1 + \dot{\theta}_1^2 a_1 c_1$$

$$= (\ddot{\theta}_1+\ddot{\theta}_2)a_2 c_{12} + \dot{\theta}_1^2 a_1 c_1 + \begin{bmatrix} a_2 s_{12} + a_1 s_1 & a_2 s_{12} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} -(\dot{\theta}_1+\dot{\theta}_2)^2 a_2 s_{12} - \dot{\theta}_1^2 a_1 s_1 \\ (\dot{\theta}_1+\dot{\theta}_2)^2 a_2 c_{12} + \dot{\theta}_1^2 a_1 c_1 \end{bmatrix} + \begin{bmatrix} a_2 c_{12} + a_1 c_1 & a_2 c_{12} \\ a_2 s_{12} + a_1 s_1 & a_2 s_{12} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

$$\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \frac{a_{22}}{a_{11}a_{22} - a_{12}a_{21}} & \frac{-a_{12}}{a_{11}a_{22} - a_{12}a_{21}} \\ \frac{-a_{21}}{a_{11}a_{22} - a_{12}a_{21}} & \frac{a_{11}}{a_{11}a_{22} - a_{12}a_{21}} \end{bmatrix} \begin{bmatrix} \ddot{x} + (\dot{\theta}_1+\dot{\theta}_2)^2 a_2 s_{12} + \dot{\theta}_1^2 a_1 s_1 \\ \ddot{y} - (\dot{\theta}_1+\dot{\theta}_2)^2 a_2 c_{12} - \dot{\theta}_1^2 a_1 c_1 \end{bmatrix}$$

```
0, sin(theta1(t)), cos(theta1(t)), 0, 0, 0,
1])&*linalg[matrix](3,1,[a1*sin(theta2(t))+a2*sin(theta2(t)+theta3(t)),
0,-a1*cos(theta2(t))-a2*cos(theta2(t)+theta3(t))+baseheigth]));
```

$$Position := \begin{bmatrix} \cos(\theta 1(t))\,(a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t))) \\ \sin(\theta 1(t))\,(a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t))) \\ -a1\,\cos(\theta 2(t)) - a2\,\cos(\theta 2(t) + \theta 3(t)) + baseheigth \end{bmatrix}$$

```
>Velocity:=map(diff,Position,t);
```

Velocity :=

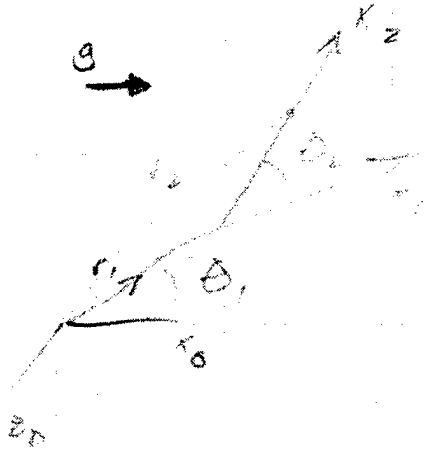
$$\left[-\sin(\theta 1(t))\left(\frac{\partial}{\partial t}\theta 1(t)\right)(a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t))) + \cos(\theta 1(t))\right.$$
$$\left.\left(a1\,\cos(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right) + a2\,\cos(\theta 2(t) + \theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right) + \left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)\right)\right]$$
$$\left[\cos(\theta 1(t))\left(\frac{\partial}{\partial t}\theta 1(t)\right)(a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t))) + \sin(\theta 1(t))\right.$$
$$\left.\left(a1\,\cos(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right) + a2\,\cos(\theta 2(t) + \theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right) + \left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)\right)\right]$$
$$\left[a1\,\sin(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right) + a2\,\sin(\theta 2(t) + \theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right) + \left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)\right]$$

```
>Acceleration:=map(diff,Velocity,t);
```

Acceleration :=

$$\left[-\cos(\theta 1(t))\left(\frac{\partial}{\partial t}\theta 1(t)\right)^2 (a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t)))\right.$$
$$-\sin(\theta 1(t))\left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right)(a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t))) - 2\sin(\theta 1(t))$$
$$\left(\frac{\partial}{\partial t}\theta 1(t)\right)$$
$$\left(a1\,\cos(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right) + a2\,\cos(\theta 2(t) + \theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right) + \left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)\right) +$$
$$\cos(\theta 1(t))\left(-a1\,\sin(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right)^2 + a1\,\cos(\theta 2(t))\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)\right.$$
$$-a2\,\sin(\theta 2(t) + \theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right) + \left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)^2$$
$$\left.\left.+ a2\,\cos(\theta 2(t) + \theta 3(t))\left(\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right) + \left(\frac{\partial^2}{\partial t^2}\theta 3(t)\right)\right)\right)\right]$$
$$\left[-\sin(\theta 1(t))\left(\frac{\partial}{\partial t}\theta 1(t)\right)^2 (a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t)))\right.$$
$$\left.+\cos(\theta 1(t))\left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right)(a1\,\sin(\theta 2(t)) + a2\,\sin(\theta 2(t) + \theta 3(t))) + 2\cos(\theta 1(t))\right)$$

$$\left(\frac{\partial}{\partial t}\theta 1(t)\right)$$
$$\left(a1\cos(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right)+a2\cos(\theta 2(t)+\theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right)+\left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)\right)+$$
$$\sin(\theta 1(t))\left(-a1\sin(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right)^2+a1\cos(\theta 2(t))\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)\right.$$
$$-a2\sin(\theta 2(t)+\theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right)+\left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)^2$$
$$\left.+a2\cos(\theta 2(t)+\theta 3(t))\left(\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)+\left(\frac{\partial^2}{\partial t^2}\theta 3(t)\right)\right)\right)\Bigg]$$
$$\left[a1\cos(\theta 2(t))\left(\frac{\partial}{\partial t}\theta 2(t)\right)^2+a1\sin(\theta 2(t))\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)\right.$$
$$+a2\cos(\theta 2(t)+\theta 3(t))\left(\left(\frac{\partial}{\partial t}\theta 2(t)\right)+\left(\frac{\partial}{\partial t}\theta 3(t)\right)\right)^2$$
$$\left.+a2\sin(\theta 2(t)+\theta 3(t))\left(\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)+\left(\frac{\partial^2}{\partial t^2}\theta 3(t)\right)\right)\right]$$



$$f_2 = m_2 g + f_{12}$$
$$= m_2 \ddot{r}_2$$

$$f_1 = m_1 g + f_{01} - f_{12}$$
$$= m_1 \ddot{r}_1$$

$$r_1 = -.0129 x_1$$

$$r_2 = l_1 x_1 + .105 x_2$$

$$n_2 = -.105 x_2 \times f_{12} + n_{12}$$
$$= I_2 \dot{\omega}_2 + \omega_2 \times I_2 \omega_2$$

$$n_1 = .0129 x_1 \times f_{01} + .0129 x_1 \times f_{12} + n_{01} - n_{12}$$
$$= I_1 \dot{\omega}_1 + \omega_1 \times I_1 \omega_1$$

$$\tau_2 = z_0 \cdot n_{12}$$

$$\tau_1 = z_0 \cdot n_{01}$$

$$\omega_1 = \dot{\theta}_1 z_0 \qquad \omega_2 = \dot{\theta}_1 z_0 + \dot{\theta}_2 z_1$$

$$\dot{\omega}_1 = \ddot{\theta}_1 z_0 \qquad \dot{\omega}_2 = (\ddot{\theta}_1 + \ddot{\theta}_2) z_0$$

$$\dot{r}_1 = -.0129 \omega_1 \times x_1$$
$$= -.013 \dot{\theta}_1 z_0 \times x_1$$
$$= -.013 \dot{\theta}_1 y_1$$

$$\ddot{r}_1 = -.013 \ddot{\theta}_1 y_1 - .013 \dot{\theta}_1 \omega_1 \times y_1$$
$$= -.013 \ddot{\theta}_1 y_1 - .013 \dot{\theta}_1 (\dot{\theta}_1 z_0) \times y_1$$
$$= -.013 \ddot{\theta}_1 y_1 + .013 \dot{\theta}_1^2 x_1$$

$$\dot{r}_2 = l_1 \omega \times x_1 + .105 \omega_2 \times x_2$$
$$= l_1 \dot{\theta}_1 z_0 \times x_1 + .105 (\dot{\theta}_1 + \dot{\theta}_2) z_0 \times x_2$$
$$= l_1 \dot{\theta}_1 y_1 + .105 (\dot{\theta}_1 + \dot{\theta}_2) y_2$$

$$\ddot{r}_2 = l_1 \ddot{\theta}_1 y_1 + l_1 \dot{\theta}_1^2 z_0 \times y_1 + .105 (\ddot{\theta}_1 + \ddot{\theta}_2) y_2$$
$$+ .105 (\dot{\theta}_1 + \dot{\theta}_2)((\dot{\theta}_1 + \dot{\theta}_2) z_0) \times y_2$$
$$= l_1 \ddot{\theta}_1 y_1 - l_1 \dot{\theta}_1^2 x_1 + .105 (\ddot{\theta}_1 + \ddot{\theta}_2) y_2 - .105 (\dot{\theta}_1 + \dot{\theta}_2)^2 x_2$$

$$f_1 = m_1(-.013\ddot{\theta}_1 x_1 + .013\dot{\theta}_1^2 x_1) = m_1 g + f_{01} - f_{12}$$

$$f_2 = m_2(l_1\ddot{\theta}_1 y_1 - l_1\dot{\theta}_1^2 x_1 + .105(\ddot{\theta}_1+\ddot{\theta}_2) y_2 - .105(\dot{\theta}_1+\dot{\theta}_2)^2 x_2)$$

$$n_1 = I_1\ddot{\theta}_1 z_0 + \dot{\theta}_1 z_0 \times I_1 \dot{\theta}_1 z_0$$
$$n_2 = I_2(\ddot{\theta}_1+\ddot{\theta}_2) z_0 + (\dot{\theta}_1+\dot{\theta}_2) z_0 \times I_2(\dot{\theta}_1+\dot{\theta}_2) z_0$$

$$\begin{aligned}
\tau_2 &= z_0 \cdot (n_2 + .105 x_2 \times f_{12}) \\
&= z_0 \cdot n_2 + z_0 \cdot (.105 x_2 \times (f_2 - m_2 g x_0)) \\
&= z_0 \cdot n_2 + .105 y_2 \cdot (f_2 - m_2 g x_0) \\
&= z_0 \cdot n_2 + .105 y_2 \cdot f_2 - .105 m_2 g \sin(\theta_1+\theta_2) \\
&= {}^0I_{2,33}(\ddot{\theta}_1+\ddot{\theta}_2) + .105 y_2 \cdot m_2(l_1\ddot{\theta}_1 y_1 - l_1\dot{\theta}_1^2 x_1 + .105(\ddot{\theta}_1+\ddot{\theta}_2) y_2 - .105(\dot{\theta}_1+\dot{\theta}_2)^2 x_2) \\
&\quad - .105 m_2 g \sin(\theta_1+\theta_2) \\
&= {}^0I_{2,33}(\ddot{\theta}_1+\ddot{\theta}_2) + .105\left[m_2 l_1 \ddot{\theta}_1 \cos\theta_2 + m_2 l_1 \dot{\theta}_1^2 \sin\theta_2 + m_2 .105(\ddot{\theta}_1+\ddot{\theta}_2) + 0\right] - G \\
&= {}^0I_{2,33}(\ddot{\theta}_1+\ddot{\theta}_2) + .105\left[m_2 l_1 \ddot{\theta}_1 \cos\theta_2 + m_2 l_1 \dot{\theta}_1^2 \sin\theta_2 + m_2 (.105)(\ddot{\theta}_1+\ddot{\theta}_2)\right] - .105 m_1 g \sin(\theta_1+\theta_2)
\end{aligned}$$

$$\begin{aligned}
\tau_1 &= z_0 \cdot (n_1 + n_{12} - .013 x_1 \times f_{01} - .013 x_1 \times f_{12}) \\
&= z_0 \cdot n_1 + z_0 \cdot n_{12} - .013 y_1 \cdot (f_{01} + f_{12})
\end{aligned}$$

$$\tau_2 = z_0 \cdot n_{12}$$

$$\begin{aligned}
\therefore \tau_1 &= z_0 \cdot n_1 + \tau_2 - .013 y_1 \cdot (f_1 - m_1 g x_0 + 2 f_{12}) \\
&= z_0 \cdot n_1 + \tau_2 - .013 y_1 \cdot f_1 + .013 m_1 g y_1 \cdot x_0 - .026 y_1 \cdot (f_2 - m_2 g x_0) \\
&= z_0 \cdot n_1 + \tau_2 - .013 x_1 \cdot f_1 - .013 m_1 g \sin\theta_1 - .026 x_1 \cdot f_2 - .026 m_2 g \sin\theta_1
\end{aligned}$$

$$z_0 \cdot n_1 = {}^0I_{1,33}\ddot{\theta}_1$$

$${}^0y_1 \cdot f_1 = {}^0y_1 \cdot m_1(-.013\ddot{\theta}_1 {}^0y_1 + .013\dot{\theta}_1^2 {}^0x_1) = -.013 m_1 \ddot{\theta}_1$$

$$\begin{aligned}
{}^0y_1 \cdot f_2 &= {}^0y_1 \cdot m_2(l_1\ddot{\theta}_1 {}^0y_1 - l_1\dot{\theta}_1^2 {}^0x_1 + .105(\ddot{\theta}_1+\ddot{\theta}_2) {}^0y_2 - .105(\dot{\theta}_1+\dot{\theta}_2)^2 {}^0x_2) \\
&= m_2 l_1 \ddot{\theta}_1 + 0 + .105 m_2(\ddot{\theta}_1+\ddot{\theta}_2)\cos\theta_2 - .105(\dot{\theta}_1+\dot{\theta}_2)^2 \sin\theta_2
\end{aligned}$$

$$\tau_1 = {}^0I_{1,33}\ddot{\theta}_1 + {}^0I_{2,33}(\ddot{\theta}_1+\ddot{\theta}_2) + .105\left[m_2 l_1 \ddot{\theta}_1 \cos\theta_2 + m_2 l_1 \dot{\theta}_1^2 \sin\theta_2 + m_2(.105)(\ddot{\theta}_1+\ddot{\theta}_2)\right]$$
$$-.105 m_2 g \sin(\theta_1+\theta_2) - .013(-.013 m_1 \ddot{\theta}_1) - .013 m_1 g \sin\theta_1$$
$$-.026\left[m_2 l_1 \ddot{\theta}_1 + .105 m_2(\ddot{\theta}_1+\ddot{\theta}_2)\cos\theta_2 - .105(\dot{\theta}_1+\dot{\theta}_2)^2 \sin\theta_2\right]$$
$$-.026 m_2 g \sin\theta_1$$

$$\tau_1 = \left[{}^0I_{1,33} + I_{2,33} + .105\left[m_2 l_1 \cos\theta_2 + m_2(.105)\right] + (-.013)^2 m_1 - .026\left[m_2 l_1 + .105 m_2\right]\right]\ddot{\theta}_1$$
$$+\left[{}^0I_{2,33} + (.105)^2 m_2 - .026(.105\cos\theta_2 m_2)\right]\ddot{\theta}_2$$
$$-.026(.105)\sin\theta_2\Big]\dot{\theta}_2^2 + \left[(-.026)(-.105)\sin\theta_2\right]\dot{\theta}_2\dot{\theta}_1 + \left[(-.026)(.105)\sin\theta_2 + (.105) m_2 l_1 \sin\theta_2\right]\dot{\theta}_1^2$$
$$-.105 m_2 g \sin(\theta_1+\theta_2) - .013 m_1 g \sin\theta_1 - .026 m_2 g \sin\theta_1$$

$$\tau_2 = \left[{}^0I_{2,33} + .105 m_2 l_1 \cos\theta_2 + (.105)^2 m_2\right]\ddot{\theta}_1 + \left[{}^0I_{2,33} + (.105)^2 m_2\right]\ddot{\theta}_2$$
$$+\left[.105 m_2 l_1 \sin\theta_2\right]\dot{\theta}_1^2 - .105 m_2 g \sin(\theta_1+\theta_2)\Big]$$

Equations based on earliest inertial parameters estimates from ProEngineer, matching hand calculations.

```
> cog1:=-.0260:
> cog2:=.105:
> z0:=linalg[matrix](3,1,[0,0,1]):
> y0:=linalg[matrix](3,1,[0,1,0]):
> y0 := matrix([[0], [1], [0]]):
> x0:=linalg[matrix](3,1,[1,0,0]):
> R01:=evalm(linalg[matrix](3,3,[cos(theta1(t)), -sin(theta1(t)), 0,
sin(theta1(t)), cos(theta1(t)), 0, 0, 0,
1])&*evalf(linalg[matrix](3,3,[1, 0, 0, 0, cos(0), -sin(0), 0, sin(0),
cos(0)]))):
> R12:=evalm(linalg[matrix](3,3,[cos(theta2(t)), -sin(theta2(t)), 0,
sin(theta2(t)), cos(theta2(t)), 0, 0, 0,
1])&*evalf(linalg[matrix](3,3,[1, 0, 0, 0, cos(0), -sin(0), 0, sin(0),
cos(0)]))):
> R02:=evalm(R01&*R12):
> R1:=evalm(cog1*R01&*x0):
> R2:=evalm(a1*R01&*x0+cog2*R02&*x0):
> R1d:=map(diff, R1, t):
> R1dd:=map(diff,R2d,t):
> R2d:=map(diff, R2, t):
> R2dd:=map(diff,R2d,t):
> omega1:=evalm(diff(theta1(t),t)*z0):
>omega2:=evalm(evalm(diff(theta1(t),t)*z0)+evalm(diff(theta2(t),t)*R01&*
z0)):
> alpha1:=map(diff,omega1,t):
```

```
> alpha2:=map(diff,omega2,t):
> g:=evalm(G*x0):
> f12:=evalm(m2*R2dd-m2*g):
>I2:=evalm(R02&*linalg[matrix](3,3,[ia2,0,0,0,ib2,0,0,0,ic2])):
> x2:=evalm(R02&*x0):
> Sx2:=linalg[matrix](3,3,[0,-x2[3,1], x2[2,1],x2[3,1],0,-x2[1,1],-
x2[2,1],x2[1,1],0]):
> evalm(omega2):
> Somega2:=linalg[matrix](3,3,[0,-omega2[3,1],
omega2[2,1],omega2[3,1],0,-omega2[1,1],-omega2[2,1],omega2[1,1],0]):
>n12:=evalm(R02&*I2&*alpha2+Somega2&*R02&*I2&*omega2+cog2*Sx2&*f12):
>torq2:=simplify(evalm(z0[1,1]*n12[1,1]+z0[2,1]*n12[2,1]+z0[3,1]*n12[3,1
])):
> evalf(torq2,3);
```

$$.105\sin(\theta 1(t))\cos(\theta 2(t))\,m2\,G + .105\cos(\theta 1(t))\sin(\theta 2(t))\,m2\,G$$
$$+ .105\cos(\theta 2(t))\,m2\,al\left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) + .105\,m2\sin(\theta 2(t))\,al\left(\frac{\partial}{\partial t}\theta 1(t)\right)^2$$
$$+ .0110\,m2\left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) + .0110\,m2\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right) + ic2\left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right) + ic2\left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right)$$

```
> Somega1:=linalg[matrix](3,3,[0,-omega1[3,1],
omega1[2,1],omega1[3,1],0,-omega1[1,1],-omega1[2,1],omega1[1,1],0]):
> evalm(omega1):
> I1:=linalg[matrix](3,3,[ia1,0,0,0,ib1,0,0,0,ic1]):
> x1:=evalm(R01&*x0):
> Sx1:=linalg[matrix](3,3,[0,-x1[3,1], x1[2,1],x1[3,1],0,-x1[1,1],-
x1[2,1],x1[1,1],0]):
> f01:=evalm(m1*R1dd-m1*g+f12):
>n01:=evalm(R01&*I1&*alpha1+Somega1&*R01&*I1&*omega1+cog1*Sx1&*f01+cog1*
Sx1&*f12+n12):
```

```
>torq1:=simplify(evalm(z0[1,1]*n01[1,1]+z0[2,1]*n01[2,1]+z0[3,1]*n01[3,1
])):
> evalf(torq1,3);
```

$$.105\, m2 \sin(\theta 2(t))\, a1 \left(\frac{\partial}{\partial t}\theta 1(t)\right)^2 + .105 \cos(\theta 2(t))\, m2\, a1 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right)$$
$$+ .00454\, m1 \left(\frac{\partial}{\partial t}\theta 1(t)\right) \sin(\theta 2(t)) \left(\frac{\partial}{\partial t}\theta 2(t)\right)$$
$$+ .0109\, m2 \left(\frac{\partial}{\partial t}\theta 1(t)\right) \sin(\theta 2(t)) \left(\frac{\partial}{\partial t}\theta 2(t)\right) - .00227\, m1 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) \cos(\theta 2(t))$$
$$- .0260\, m1\, a1 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) - .0520\, m2\, a1 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) + .0110\, m2 \left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)$$
$$+ .0110\, m2 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) + .00227\, m1 \sin(\theta 2(t)) \left(\frac{\partial}{\partial t}\theta 2(t)\right)^2$$
$$- .00546\, m2 \cos(\theta 2(t)) \left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right) + .00546\, m2 \sin(\theta 2(t)) \left(\frac{\partial}{\partial t}\theta 2(t)\right)^2$$
$$+ .00546\, m2 \left(\frac{\partial}{\partial t}\theta 1(t)\right)^2 \sin(\theta 2(t)) - .00546\, m2 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right) \cos(\theta 2(t))$$
$$+ .00227\, m1 \left(\frac{\partial}{\partial t}\theta 1(t)\right)^2 \sin(\theta 2(t)) - .00227\, m1 \cos(\theta 2(t)) \left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right)$$
$$+ .105 \sin(\theta 1(t)) \cos(\theta 2(t))\, m2\, G + .105 \cos(\theta 1(t)) \sin(\theta 2(t))\, m2\, G$$
$$- .0520 \sin(\theta 1(t))\, m2\, G - .0260 \sin(\theta 1(t))\, m1\, G + ic2 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right)$$
$$+ ic2 \left(\frac{\partial^2}{\partial t^2}\theta 2(t)\right) + ic1 \left(\frac{\partial^2}{\partial t^2}\theta 1(t)\right)$$

For current parameters:

```
> cog1:=.1135:
> cog2:=.08735:
> evalf(torq2,3);
```

$$.00763\ m2\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+.00763\ m2\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)+.0874\sin(\theta1(t))\cos(\theta2(t))\ m2\ G$$
$$+.0874\cos(\theta1(t))\sin(\theta2(t))\ m2\ G+.0874\ m2\sin(\theta2(t))\ a1\left(\frac{\partial}{\partial t}\theta1(t)\right)^2$$
$$+.0874\cos(\theta2(t))\ m2\ a1\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+ic2\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+ic2\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)$$

```
> evalf(torq1,3);
```

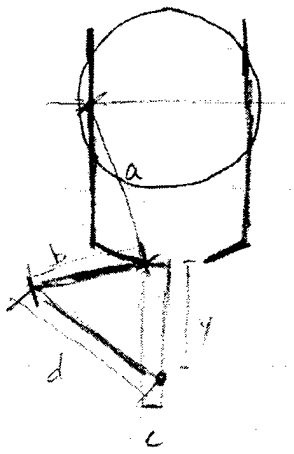
$$ic1\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+.0874\sin(\theta1(t))\cos(\theta2(t))\ m2\ G+.114\sin(\theta1(t))\ m1\ G$$
$$+.227\sin(\theta1(t))\ m2\ G+.0874\cos(\theta1(t))\sin(\theta2(t))\ m2\ G+ic2\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)$$
$$+.0874\ m2\sin(\theta2(t))\ a1\left(\frac{\partial}{\partial t}\theta1(t)\right)^2+.0874\cos(\theta2(t))\ m2\ a1\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)$$
$$+.227\ m2\ a1\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+.00763\ m2\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)$$
$$+.00991\ m1\cos(\theta2(t))\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)-.0198\ m1\left(\frac{\partial}{\partial t}\theta1(t)\right)\sin(\theta2(t))\left(\frac{\partial}{\partial t}\theta2(t)\right)$$
$$-.00991\ m1\sin(\theta2(t))\left(\frac{\partial}{\partial t}\theta2(t)\right)^2-.00991\ m1\left(\frac{\partial}{\partial t}\theta1(t)\right)^2\sin(\theta2(t))$$
$$+.0198\ m2\cos(\theta2(t))\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)-.0198\ m2\sin(\theta2(t))\left(\frac{\partial}{\partial t}\theta2(t)\right)^2$$
$$-.0198\ m2\left(\frac{\partial}{\partial t}\theta1(t)\right)^2\sin(\theta2(t))+.0198\ m2\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)\cos(\theta2(t))$$
$$+.00991\ m1\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)\cos(\theta2(t))+.114\ m1\ a1\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)$$
$$+.00763\ m2\left(\frac{\partial^2}{\partial t^2}\theta1(t)\right)+ic2\left(\frac{\partial^2}{\partial t^2}\theta2(t)\right)$$
$$-.0397\ m2\left(\frac{\partial}{\partial t}\theta1(t)\right)\sin(\theta2(t))\left(\frac{\partial}{\partial t}\theta2(t)\right)$$

Appendix C Hand Motion Calculations

Appendix C contains Maple calculations and hand drawn sketches in the rear of the appendix to illustrate variables' meanings.

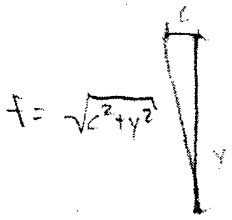
Appendix C Sketches

#1

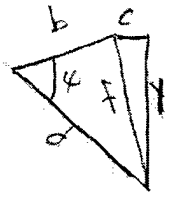


Objectives:
- find best one inch segment at y for max mechanical advantage during motion
- establish available forces for holding the ball

#2



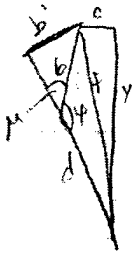
$f = \sqrt{c^2 + y^2}$



#3

$\psi = \cos^{-1}\left(\frac{f^2 - b^2 - d^2}{-2bd}\right)$

#4



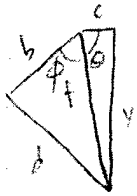
$\mu = \pi - \psi$

$b' = b\sin(\mu)$ = effective moment arm

#5

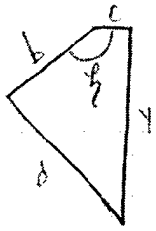
$\phi = \cos^{-1}\left(\frac{d^2 - b^2 - f^2}{-2bf}\right)$

#6



$\theta = \operatorname{atan}\left(\frac{y}{c}\right)$

#7



$\xi = \phi + \theta$

the finger is connected to b, so $\Delta\xi$ shows the angular movement the finger will experience

The purpose of this Appendix is to display the calculations used in designing the hand. There were a few things that needed to be determined to validate the idea of the solenoid acuated hand:

1) Can the solenoid move the fingers and opposing spring fast enough to release the ball without interfering with its trajectory?
2) Will the solenoid have sufficient strength to hold the ball without letting it slip?
3) Will the one-inch of pull move the fingers far enough to clear them out of the ball's path?

Section 4.7 discusses these questions and the following material supports the discussion.

Attached are sketches, which illustrate the meaning of the variables.

See Sketch #1

```
length from pivot to center of hole for ball
> a:=1.5:
length of moment arm to which solenoid connection bar attaches
> b:=1:
distance pivot is offset from the connection bar's pivot at the cable
connection
> c:=0.25:
length from moment arm connection to to cable attachment - ie the
length of the connection bar
> d:=1.5:
```

See Sketch #2

```
length between pivots
> f:=(c^2+y^2)^.5;
>
```

$$f := (.0625 + y^2)^{.5}$$

See Sketch #3

```
angle between connection bar and moment arm
> psi:=arccos((f^2-b^2-d^2)/(-2*b*d));
```

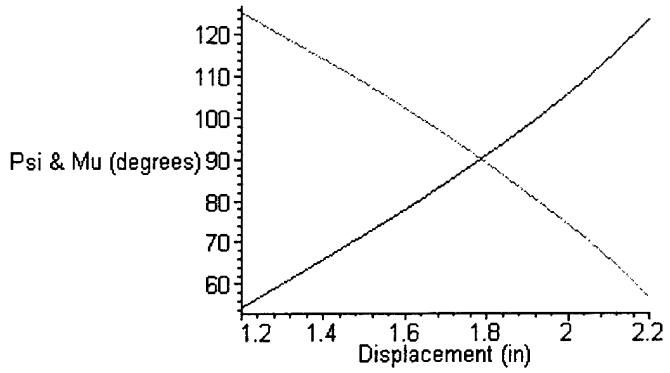
$$\psi := \pi - \arccos(.3333333334\ (.0625 + y^2)^{1.0} - 1.083333334)$$

See Sketch #4

```
angle within the right triangle that has the effective moment arm as
the base
> mu:=Pi-psi;
```

$$\mu := \arccos(.3333333334\ (.0625 + y^2)^{1.0} - 1.083333334)$$

```
the angles the connection bar makes with the moment arm at the pivot
wrt y - linear displacement
> plot([psi*180/Pi, mu*180/Pi],y=1.2..2.2,labels=["Displacement
(in)","Psi & Mu (degrees)"]);
```

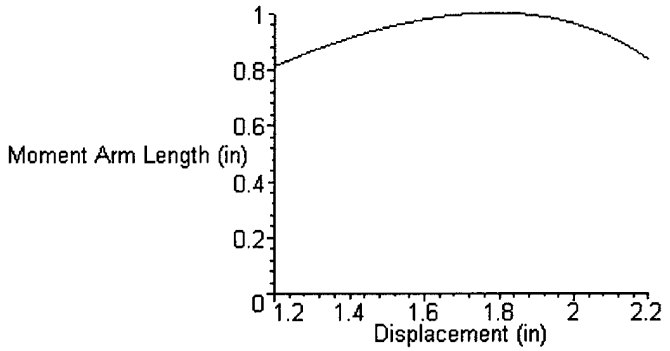


the effective moment arm wrt y - linear displacement

```
>bprime:=b*sin(mu);
```

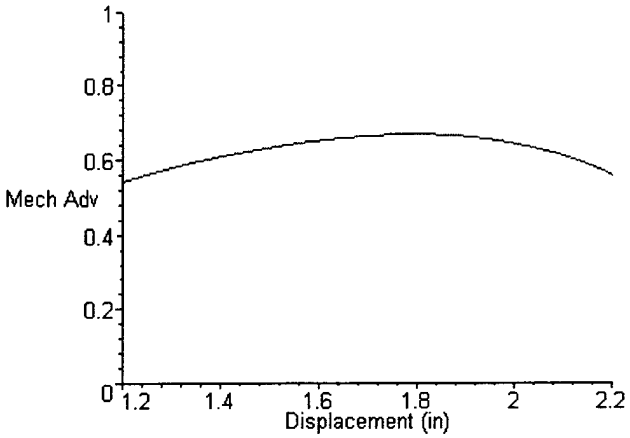
$$bprime := \sqrt{1 - (.3333333334\,(.0625 + y^2)^{1.0} - 1.083333334)^2}$$

```
>plot(bprime,y=1.2..2.2,0..1,labels=["Displacement (in)","Moment Arm
Length (in)"]);
```



Mechanical Advantage

```
>plot(bprime/a,y=1.2..2.2,0..1,labels=["Displacement (in)","Mech
Adv"]);
```




```
>Xvalues:=[0,.125,.25,.375,.5,.6125,.75,.8725,1];
```

$$Xvalues := [0, .125, .25, .375, .5, .6125, .75, .8725, 1]$$

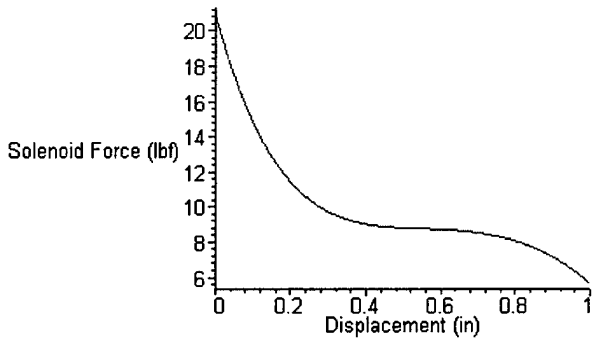
```
>Yvalues:=[336,220,168,144,142,140,132,120,90];
```

$$Yvalues := [336, 220, 168, 144, 142, 140, 132, 120, 90]$$

```
>with(stats):
>solenoidforce:=fit[leastsquare[[x,y],
y=a1*x^5+a2*x^4+a3*x^3+a4*x^2+a5*x+a6, {a1,a2,a3,a4,a5,a6}]]([Xvalues,
Yvalues]);
```

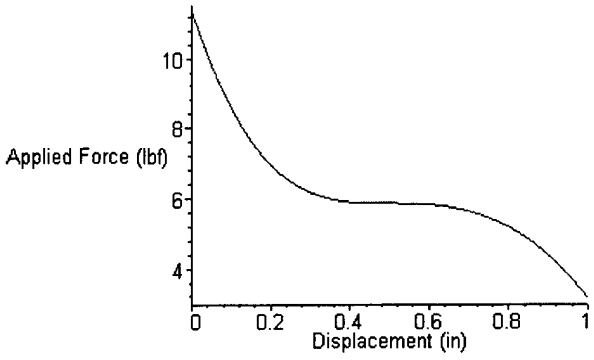
$$solenoidforce := y = -650.9492847\,x^5 + 2395.128961\,x^4 - 3916.788773\,x^3 + 3192.841030\,x^2 - 1266.030358\,x + 335.9386351$$

```
>plot((-650.9492847*x^5+2395.128961*x^4-
3916.788773*x^3+3192.841030*x^2-
1266.030358*x+335.9386351)/16,x=0..1,labels=["Displacement
(in)","Solenoid Force (lbf)"]);
```



Force Available wrt linear displacement of the solenoid

```
>plot((-650.9492847*y^5+2395.128961*y^4-
3916.788773*y^3+3192.841030*y^2-1266.030358*y+335.9386351)*(sqrt(1-
(.3333333334*(.625e-1+(y+1.2)^2)^1.0-
1.083333334)^2))/(a*16),y=0..1,labels=["Displacement (in)","Applied
Force (lbf)"]);
```



See Sketch #5

```
> phi:=arccos((d^2-b^2-f^2)/(-2*b*f));
```

$$\phi := \pi - \arccos\left(\frac{1}{2}\,\frac{1.25 - (.0625 + y^2)^{1.0}}{(.0625 + y^2)^{.5}}\right)$$

See Sketch #6

```
> theta:=arctan(y/c);
```

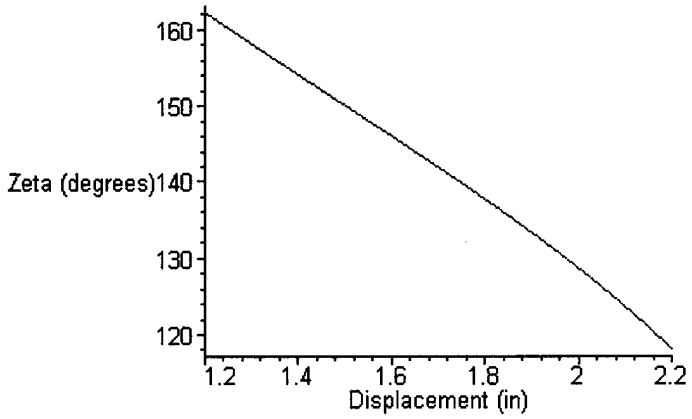
$$\theta := \arctan(4.000000000\ y)$$

See Sketch #7

```
> zeta:=phi+theta;
```

$$\zeta := \pi - \arccos\left(\frac{1}{2}\,\frac{1.25 - (.0625 + y^2)^{1.0}}{(.0625 + y^2)^{.5}}\right) + \arctan(4.000000000\ y)$$

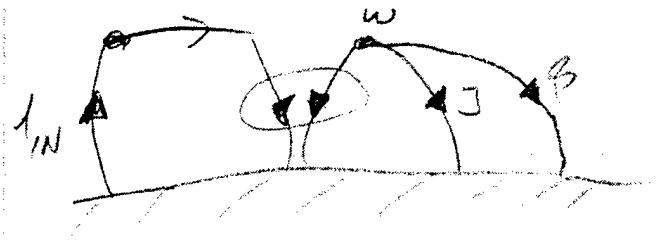
```
angular displacement wrt to linear displacement
> plot(zeta*180/Pi,y=1.2..2.2,labels=["Displacement (in)","Zeta
(degrees)"]);
```



>

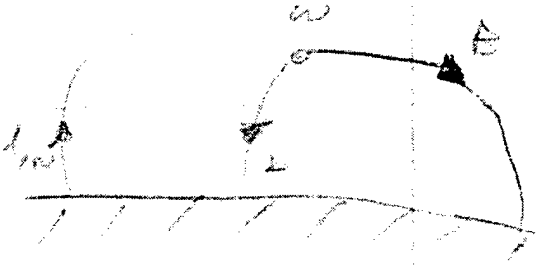
Appendix D Tree/CoTree Derivations

This appendix contains both the second order and third order system derivations of the motor, ignoring inductance and including inductance respectively.



Tree

Co-Tree



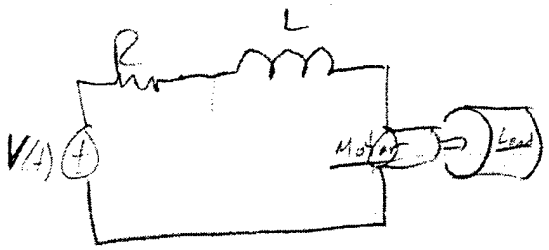
$V_L = R i_R$	$i_R = i_{IN}$	$V_R = R i_{IN}$
$V_L = -k_e \omega_b$	$\omega_R = \omega_J$	$V_L = -k_e \omega_J$
$T_b = k_T i_a$	$i_a = i_R$	$T_b = k_T i_{IN}$
$\dot{\omega}_J = \frac{1}{J} T_J$	$T_J = -T_b - T_B$	$\dot{\omega}_J = \frac{1}{J}(k_T i_{IN} - B\omega_J)$
$T_B = B\omega_B$	$\omega_B = \omega_J$	$T_B = B\omega_J$

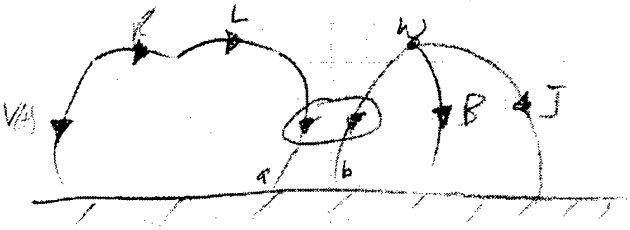
$$\dot{\omega}_J = \frac{1}{J}(-k_T i_{IN} - B\omega_J) = -\frac{k_T}{J} i_{IN} - \frac{B}{J}\omega_J$$

$$\begin{bmatrix} \dot{\theta}_J \\ \dot{\omega}_J \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} \theta_J \\ \omega_J \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{k_T}{J} \end{bmatrix} i_{IN}$$

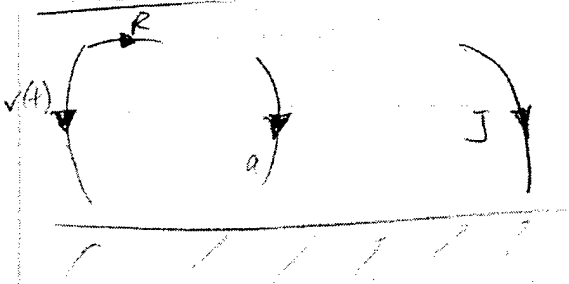
Base Motor	Link 1 Motor	Link 2 Motor
$k_t = .22915 \frac{N\cdot m}{A}$	$k_t = .22365$	$k_t = .50705$
$B = .0004 \frac{N\cdot m}{rad/s}$	$B = .0004$	$B = .0002$
$J \approx .0002^{*} \; kg\cdot m^2$	$J = .0001597$	$J = .0001807$

* since this parameter is not critical and it was difficult to access it was estimated, based on the other motors' values & the fact that the base motor was turning a relatively large steel mass - the base axle.

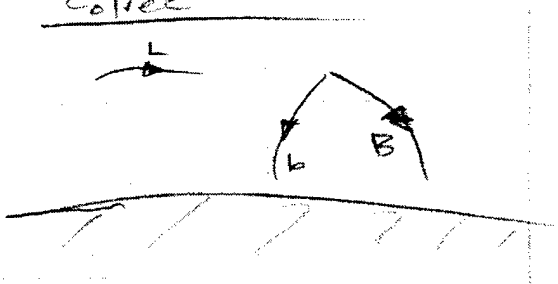




Tree



Co Tree



State Vars: ω_J, i_L

$V_R = R i_R$	$i_R = i_L$	$V_R = R i_L$
$V_a = -k_e \omega_b$	$\omega_b = \omega_J$	$V_a = -k_e \omega_J$
$\dot{\omega}_J = \frac{1}{J} T_J$	$T_J = -T_B - T_b$	$\dot{\omega}_J = -\frac{1}{J}(T_B + T_b) = -\frac{1}{J}(B\omega_J + \frac{1}{k_t} i_L)$
$\dot{i}_L = \frac{1}{L} V_L$	$V_L = v(t) - V_R - V_a$	$\dot{i}_L = \frac{1}{L}(v(t) - V_R - V_a)$
$T_B = B \omega_B$	$\omega_B = \omega_J$	$T_B = B \omega_J$
$T_b = \frac{1}{k_t} i_a$	$i_a = i_L$	$T_b = \frac{1}{k_t} i_L$

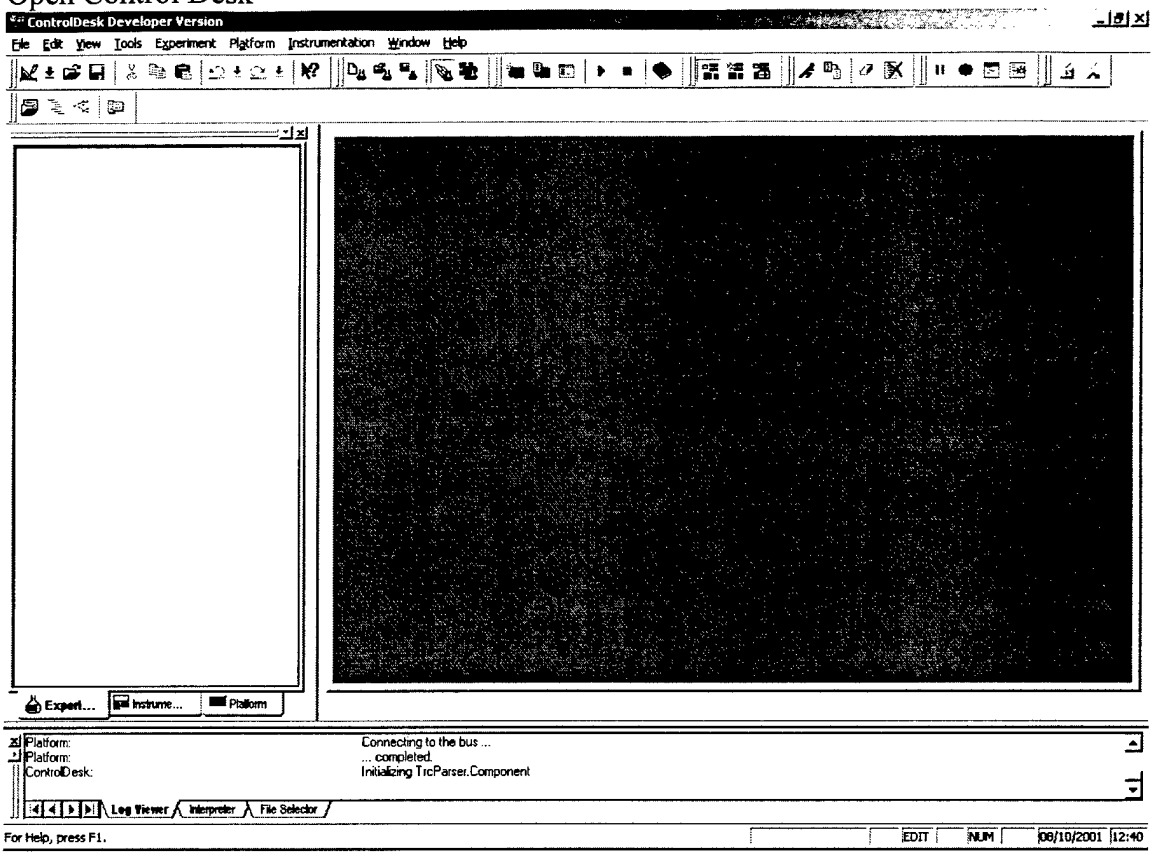
$$\dot{\omega}_J = -\frac{1}{J}\left(B\omega_J + \frac{1}{k_t} i_L\right)$$
$$\dot{i}_L = \frac{1}{L}\left(v(t) - R i_L + k_e \omega_J\right)$$
$$\dot{\theta}_J = \omega_J$$

$$\begin{bmatrix} \dot{\omega}_J \\ \dot{i}_L \\ \dot{\theta}_J \end{bmatrix} = \begin{bmatrix} -\frac{B}{J} & -\frac{1}{k_t J} & 0 \\ \frac{k_e}{L} & -\frac{R}{L} & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \omega_J \\ i_L \\ \theta_J \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \\ 0 \end{bmatrix} v(t)$$

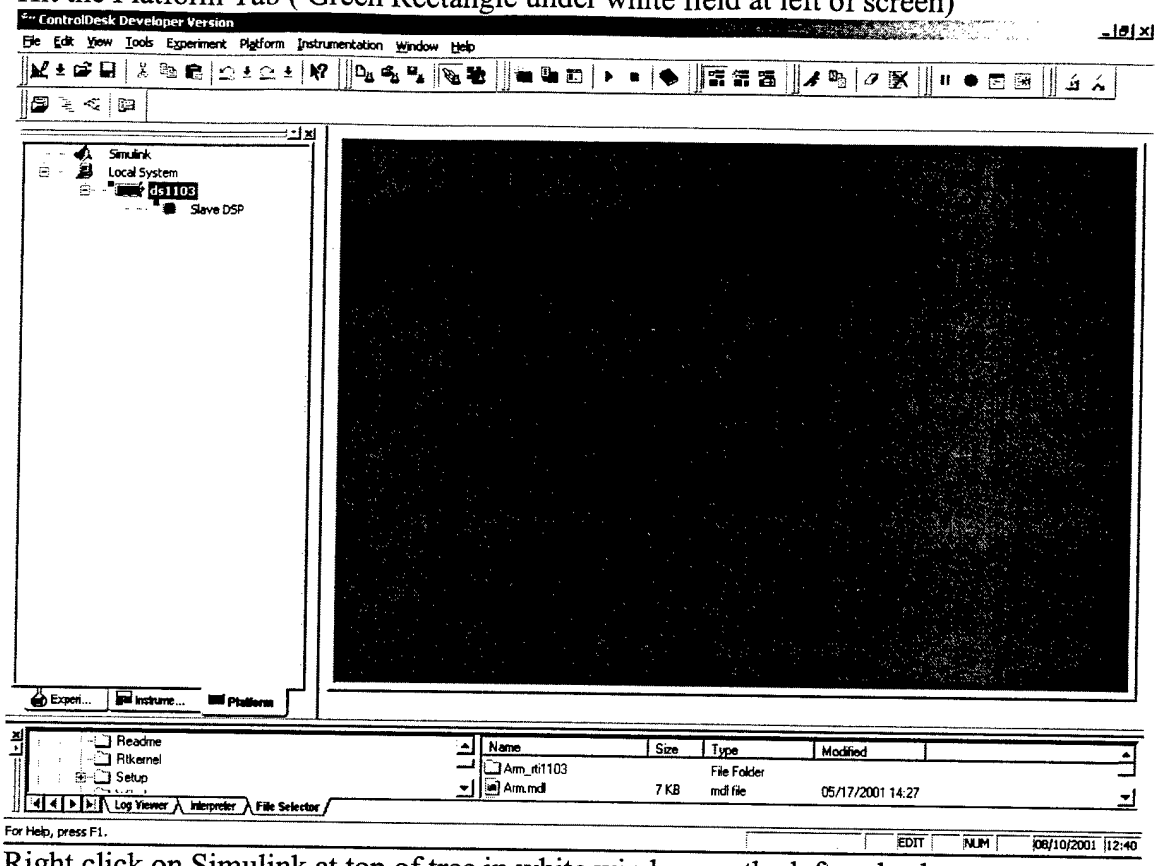
Appendix E dSpace Procedures and Simulink Code

Appendix E dSpace Procedures and Simulink Code

1. Open Control Desk

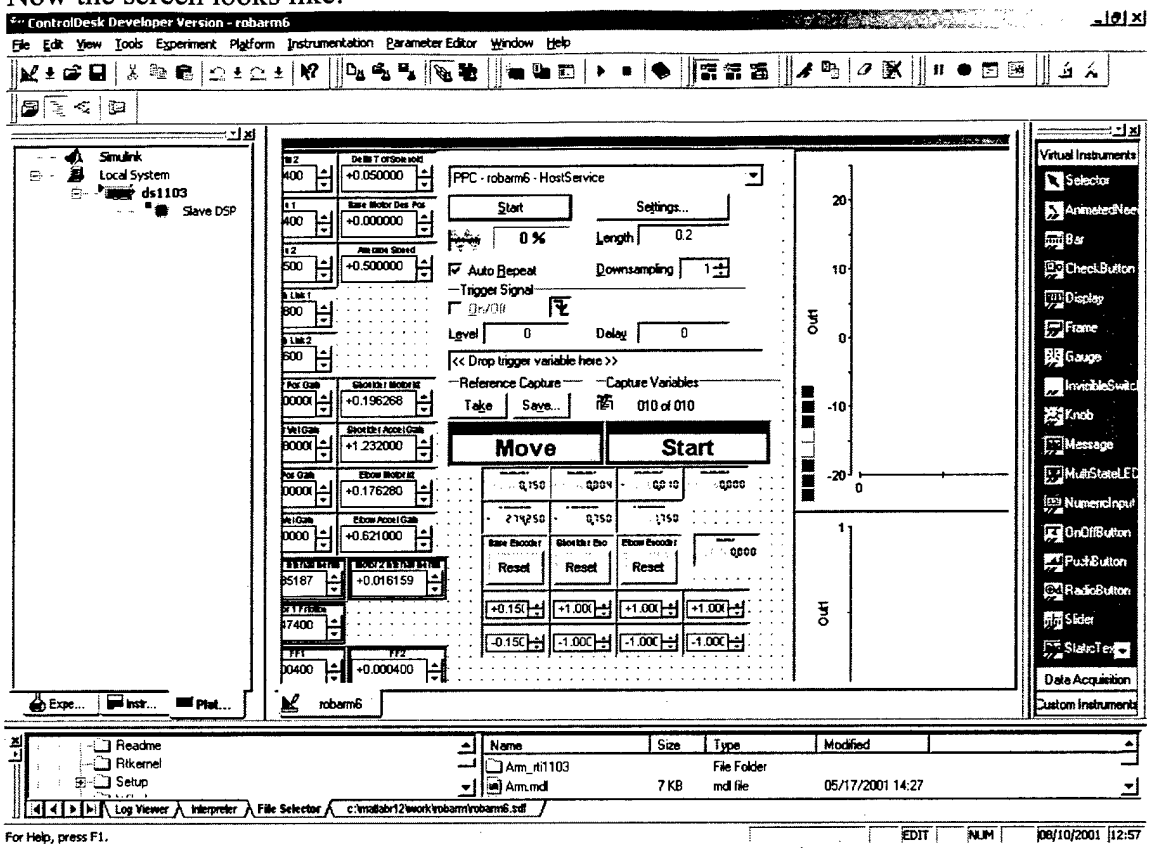


2. Hit the Platform Tab (Green Rectangle under white field at left of screen)



3. Right click on Simulink at top of tree in white window on the left and select "Open Matlab" Matlab will automatically load up.
4. In Matlab, change the working directory from c:\matlabr12\bin\win32 to » c:\matlabr12\work\RobArm
5. We have to establish certain variables as global variables, so open datainit.m for editing, type "edit datainit" and the Matlab editor will open up datainit.m, there is a whole list of global variables that must be copied into the workspace, (to be truly global, they must be declared global in the workspace and in the m file). So highlight all lines declaring global variables, copy and paste into the command line, execute, then type "datainit;" and execute to initialize the variables.
6. Go back to the Matlab main window and open Simulink, then open robarm6.mdl.
7. To load the code onto the dSpace processor, choose from the menu: Tools, Real-Time Workshop, Build Model.
8. Once loaded onto the dSpace processor, then the Control Desk GUI must be loaded, so return to the Control Desk Window and open "c:\DSPACE\work\RobArm\robarm6.lay".
9. When prompted choose to load data connections and all the GUI buttons will be tied to their associated variables in the Simulink Model.

10. Now the screen looks like:

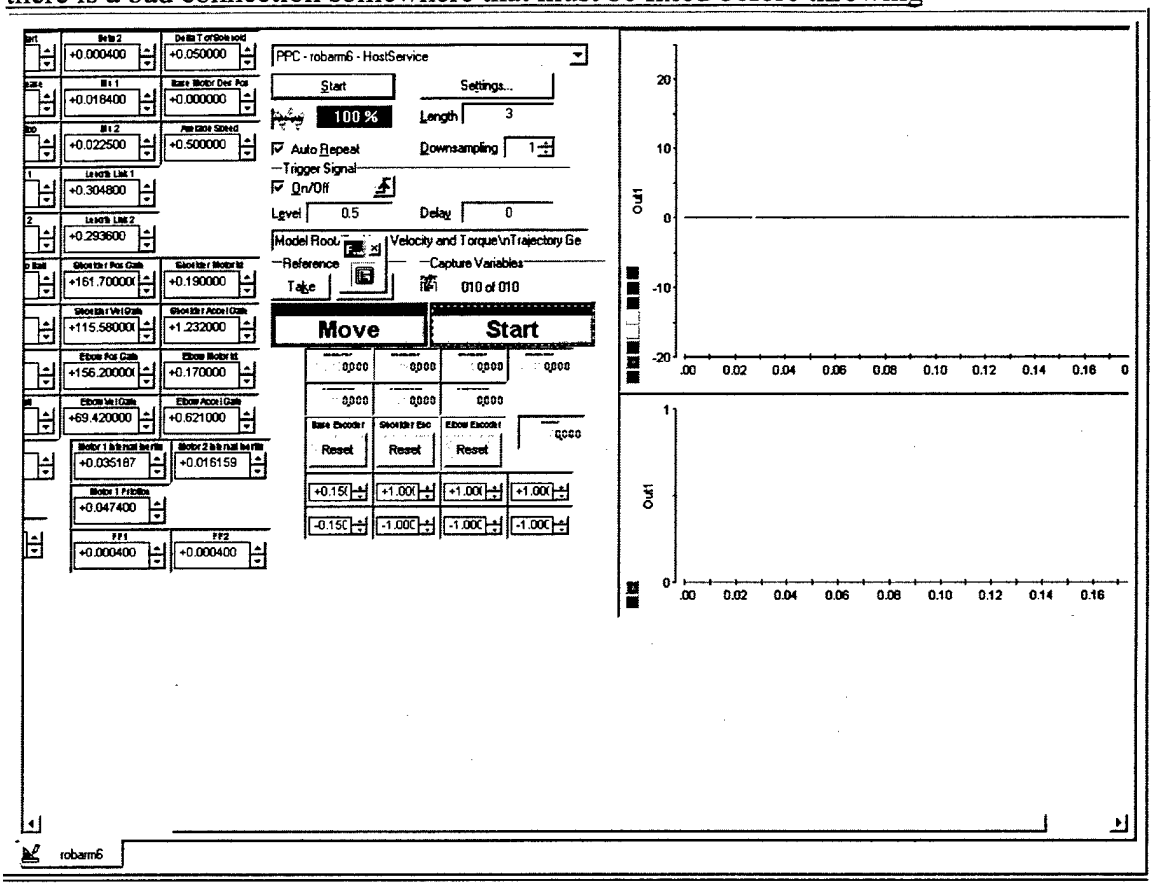


At the top of the screen there are three similar looking buttons, directly to the right of a red book, of these three the left is depressed in the picture. This designates the GUI's editing mode, the middle designates test mode and the right button is animation mode. To run the robot, press the animation mode button.

11. The default data capture mode starts with the animation for 0.02s and repeats. I usually push the stop button directly under "PPD - robarm6 – Host Service".
12. Next I right click on the red and white Start button and choose "Highlight Variables". This then shows a red chain link in the bottom right window beside the variable that the chosen button is connected to. Now I now that the Start button uses the variable "Trigger". To setup the data capture, click and hold the grey block that says value beside the red chain link and drag it up to the "<<Drop trigger variable here>>" window above. This tells the data capture what variable to use as trigger.
13. Now to configure the capture. I hit Settings, and change the attributes:
    a. I turn off the auto start with animation
    b. I change the length to 3 seconds
    c. I then hit the trigger tab and choose:
        i. "Trigger on Signal"
        ii. the Rising Edge Trigger button
        iii. a zero second delay
        iv. a level of 0.5, hit apply and exit
14. Now I usually like to get a bigger GUI, so I hit View and FullScreen

Pre-Throw Diagnostics

1. I move the base back and forth to make sure the computer is getting the position information. I move the shoulder and elbow individually too. <u>If any of these fail, there is a bad connection somewhere that must be fixed before throwing</u>



2. I set all joints at the zero position I want them to have and hit the reset buttons for each joint's encoder. Make sure that all reset buttons are released and not depressed before continuing.

* Note that in the DSpace GUI for some reason I don't understand, when a button is first hit, a dotted line appears around it and it isn't really depressed fully, when clicked again, then it actually executes, this proves somewhat annoying in certain circumstances. For example, the first time the start or move buttons are hit to engage a motion, the first value of the array is sent to the controller, but the successive ones aren't for some reason, until the button is hit again and released. Every successive use of that button without using other buttons in between uses will execute after the first click. You'll figure it out after playing with it for a while.

All editing of the GUI must be done in Editing Mode, just right click to get all of the buttons parameters for modification, etc.

To capture data, press the data capture (grey) start button at the top and then hit start, let the robot execute the throw and hit save, it'll let you choose where and by what name you save your data.

*** Note ***
Never hit the Start button after the robot arm has begun a throwing motion until it has finished the throw and returned to its starting position.

To use the Move button, first choose a desired base angle position, then hit the move button twice (once to highlight it and a second time to engage the movement), the robot will move to the desired position.
**** After the robot has finished its base motor movement, never hit the Move button again without changing the desired position, it will wind up. It's a flaw, but dSpace isn't the platform that'll be used in the schools, so I didn't work the bug out.****

I put this in last to make sure the previous sections were read first. The obvious part is that the power must be on, the less obvious, but simple point I wanted to cover is that the kill switch connects the power, so the robot will not move unless the kill switch is depressed.

Other safety features:

All electrical components are protected by the lexan covers
A red LED shows when there is voltage across the capacitor: Never open the case when the power is or when the capacitor LED indicator is lit. First turn off the power, then push the discharge button until the LED turns off, then it is safe to remove the lexan covers and work with the electrical components. ***** DO NOT PUSH THE DISCHARGE BUTTON WHILE THE POWER IS ON, IT WILL FRY - IT'S ONLY SAFE FOR DISCHARGING THE CAPACITOR, NOT FOR DISSIPATING THE POWERGRID****

```
/*
 * sfuntmpl_basic.c: Basic 'C' template for a level 2 S-function.
 *
 *  -------------------------------------------------------------------------↙
--
 *  | See matlabroot/simulink/src/sfuntmpl_doc.c for a more detailed templa↙
te |
 *  -------------------------------------------------------------------------↙
--
 *
 * Copyright 1990-2000 The MathWorks, Inc.
 * $Revision: 1.24 $
 */


/*
 * You must specify the S_FUNCTION_NAME as the name of your S-function
 * (i.e. replace sfuntmpl_basic with the name of your S-function).
 */

#define S_FUNCTION_NAME  test7b
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
#include "math.h"



/* Error handling
 * --------------
 *
 * You should use the following technique to report errors encountered with↙
in
 * an S-function:
 *
 *       ssSetErrorStatus(S,"Error encountered due to ...");
 *       return;
 *
 * Note that the 2nd argument to ssSetErrorStatus must be persistent memory↙

 * It cannot be a local variable. For example the following will cause
 * unpredictable errors:
 *
 *      mdlOutputs()
 *      {
```

Name


```
 *          char msg[256];         {ILLEGAL: to fix use "static char msg[256↙
];"}
 *          sprintf(msg,"Error due to %s", string);
 *          ssSetErrorStatus(S,msg);
 *          return;
 *      }
 *
 * See matlabroot/simulink/src/sfuntmpl_doc.c for more details.
 */

/*====================*
 * S-function methods *
 *====================*/

/* Function: mdlInitializeSizes ===========================================↙
====
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    /* See sfuntmpl_doc.c for more details on the macros below */

    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 32);
    ssSetInputPortRequiredContiguous(S, 0, true); /*direct input signal acc↙
ess*/
    ssSetInputPortDirectFeedThrough(S, 0, 30);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 10);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
```

input port dimensions

# of inputs used to calculate outputs

output port dimensions

```
    ssSetOptions(S, 0);
}



/* Function: mdlInitializeSampleTimes =========================================↙
====
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);

}



#undef MDL_INITIALIZE_CONDITIONS   /* Change to #undef to remove function *↙
/
#if defined(MDL_INITIALIZE_CONDITIONS)
  /* Function: mdlInitializeConditions ========================================↙
====
   * Abstract:
   *    In this function, you should initialize the continuous and discrete
   *    states for your S-function block.  The initial states are placed
   *    in the state vector, ssGetContStates(S) or ssGetRealDiscStates(S).
   *    You can also perform any other initialization activities that your
   *    S-function may require. Note, this routine will be called at the
   *    start of simulation and if it is present in an enabled subsystem
   *    configured to reset states, it will be call when the enabled subsys↙
tem
   *    restarts execution to reset the states.
   */
  static void mdlInitializeConditions(SimStruct *S)
  {


  /*
      float       th11,th12,th13,th14,th21,th22,th23,th24,t1poly1,t1poly2,t↙
1poly3,t1poly4,t1poly5,t1poly6,t1poly7,t1poly8,t1poly9,t1poly10,t1poly11,t2↙
poly1,t2poly2,t2poly3,t2poly4,t2poly5,t2poly6,t2poly7,t2poly8,t2poly9,t2pol↙
y10,t2poly11;

      FILE *fid;
```



```
      fid = fopen("trajcoef.txt","r");
      fscanf(fid,"%12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %↙
12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f↙
 %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f",th11,th12,↙
th13,th14,th21,th22,th23,th24,t1poly1,t1poly2,t1poly3,t1poly4,t1poly5,t1pol↙
y6,t1poly7,t1poly8,t1poly9,t1poly10,t1poly11,t2poly1,t2poly2,t2poly3,t2poly↙
4,t2poly5,t2poly6,t2poly7,t2poly8,t2poly9,t2poly10,t2poly11);

      printf("%12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8↙
f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12↙
.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f\n",th11,th12,th↙
13,th14,th21,th22,th23,th24,t1poly1,t1poly2,t1poly3,t1poly4,t1poly5,t1poly6↙
,t1poly7,t1poly8,t1poly9,t1poly10,t1poly11,t2poly1,t2poly2,t2poly3,t2poly4,↙
t2poly5,t2poly6,t2poly7,t2poly8,t2poly9,t2poly10,t2poly11);

      fclose(fid);
      fid = fopen("trajresp.txt","w");
      fprintf(fid,"%12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f ↙
%12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8↙
f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f %12.8f",th11,th12↙
,th13,th14,th21,th22,th23,th24,t1poly1,t1poly2,t1poly3,t1poly4,t1poly5,t1po↙
ly6,t1poly7,t1poly8,t1poly9,t1poly10,t1poly11,t2poly1,t2poly2,t2poly3,t2pol↙
y4,t2poly5,t2poly6,t2poly7,t2poly8,t2poly9,t2poly10,t2poly11);
      fclose(fid);*/

  }
#endif /* MDL_INITIALIZE_CONDITIONS */



#undef MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
  /* Function: mdlStart =======================================================↙
====
   * Abstract:
   *    This function is called once at start of model execution. If you
   *    have states that should be initialized once, this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)
  {
  }
#endif /*  MDL_START */



/* Function: mdlOutputs =======================================================↙
====
 * Abstract:
```

```
 *    In this function, you compute the outputs of your S-function
 *    block. Generally outputs are placed in the output vector, ssGetY(S).
 */

    real_T       st=0;
    real_T       xt=0;
    real_T       t=0;
    real_T       t3=0;
    real_T       movet=0;

static void mdlOutputs(SimStruct *S, int_T tid)
{

    const real_T *u = (const real_T*) ssGetInputPortSignal(S,0);
    real_T       *y = ssGetOutputPortSignal(S,0);


    /* variables read in through the input u */

    real_T       duration, th1s, th1r, th1f, th2s, th2r, th2f, w1s, w1r, w1↙
f, w2s, w2r, w2f, m1, m2, m2noball, I133, I233, I233noball, beta1, beta2, m↙
u1, mu2, l1, l2, deltat;

    /* variables used within the code for computational ease */

    real_T       A11, A12, A21, A22, B1, B2, H1, H2, G1, G2, C1, C2, t1, t2↙
, w1, w2, a1, a2, g, tq1, tq2, th11, th12, th13, th14, th21, th22, th23, th↙
24, th1ra,th1rb,th2ra,th2rb;

    /* variables used for the movement of the Base Motor */

    real_T      th31,th32,th33,th34;

        y[0]=0;
        y[1]=0;
        y[2]=0;
        y[3]=0;
        y[4]=0;
        y[5]=0;
        y[6]=0;
        y[7]=0;
        y[8]=0;
        y[9]=t3;

duration=u[0];
th1s=u[1];
th1r=u[2];
th1f=u[3];
```

- variables that need to keep their values through time step transitions

initialize variables

Control Code


```
th2s=u[4];
th2r=u[5];
th2f=u[6];
w1s=u[7];
w1r=u[8];
w1f=u[9];
w2s=u[10];
w2r=u[11];
w2f=u[12];
m1=u[13];
m2=u[14];
m2noball=u[15];
I133=u[16];
I233=u[17];
I233noball=u[18];
beta1=u[19];
beta2=u[20];
mu1=u[21];
mu2=u[22];
l1=u[23];
l2=u[24];
deltat=u[25];
g=9.8;
t1=0;
t2=0;
w1=0;
w2=0;
a1=0;
a2=0;
A11=0;
A12=0;
A21=0;
A22=0;
B1=0;
B2=0;
H1=0;
H2=0;
G1=0;
G2=0;
C1=0;
C2=0;
tq1=0;
tq2=0;
th1ra=th1r-.5*deltat*w1r;
th1rb=th1r+.5*deltat*w1r;
th2ra=th2r-.5*deltat*w2r;
th2rb=th2r+.5*deltat*w2r;

/*This conditional checks for the logical start command, which can only ini↙
```

load From input ports

```
tiate a movement after the program has been running
  for more than ~2 second.  If the logical start has been strobed, then at ↙
that instant the variable st (or xt for the Motor 3 command), is given the ↙
current
  clock time or start time (st).  Therefore the time variable, as seen from↙
 within the s-function trajectory generator, is reset
  and will execute a new throwing motion as commanded by the variables cont↙
rolled from the D-Space Control Desk. */


if(u[26]>2*duration+deltat)
    {
    if(u[27]>.5)
        {
        st=u[26];
        }

    if(u[28]>.5)
        {
        xt=u[26];
        movet=abs(u[30]-u[29])/u[31];
        }

    t=u[26]-xt;

    if((u[30]-u[29])!=0)
        {
        if(t>=0&&t<=movet)
            {
            th31 = u[29];
            th32 = 0;
            th33 =  (3 * (u[30]-u[29]))/(movet*movet);
            th34 = (-2 * (u[30]-u[29]))/(movet*movet*movet);

            t3 =  th31 + t * (th32 + t * (th33 + t * th34));
            }
        }

   t=u[26]-st;

   if( t >= (duration+.45*deltat) && t <= (2*duration+deltat ) ) y[8]=5;


   if( t <= (duration) && t >= 0 )
       {

       /* Generate coefficients of cubic polynomial.*/
```

sets this start time of the movement

sets the base motor start time

base motor & trajectory generator

link 1 & link 2 trajectory generator


```
       th11 = th1s;
       th12 = w1s;
       th13 =  (3 * (th1ra-th1s) - duration * (2 * w1s + w1r)) / (duration*↙
duration);
       th14 = (-2 * (th1ra-th1s) + duration * (w1s + w1r)) / (duration*dura↙
tion*duration);

       th21 = th2s;
       th22 = w2s;
       th23 =  (3 * (th2ra-th2s) - duration * (2 * w2s + w2r)) / (duration*↙
duration);
       th24 = (-2 * (th2ra-th2s) + duration * (w2s + w2r)) / (duration*dura↙
tion*duration);

       /* Generate position, velocity and acceleration at this time t */

       t1 =  th11 + t * (th12 + t * (th13 + t * th14));
       t2 =  th21 + t * (th22 + t * (th23 + t * th24));

       w1 = th12+t*(2*th13+3*t*th14);
       w2 = th22+t*(2*th23+3*t*th24);

       a1 = 2*th13+6*th14*t;
       a2 = 2*th23+6*th24*t;



       /* Calculate torque components */


       A11=I133+I233-.00273*m2*cos(t2)-.0260*m2*l1+.0110*m2+.105*cos(t2)*m2↙
*l1;
       A12=I233+.0110*m2-.00273*m2*cos(t2);
       A21=I233+.105*m2*l1*cos(t2)+.0110*m2;
       A22=I233+.0110*m2;

       H1=.0027*m2*sin(t2)*w2*w2+.00546*m2*sin(t2)*w1*w2+(.00273*m2*sin(t2)↙
+.105*m2*l1*sin(t2))*w1*w1;
       H2=.105*m2*l1*sin(t2)*w1*w1;

       G1=.105*m2*g*sin(t1+t2)-(.013*m1+.026*m2)*g*sin(t1);
       G2=.105*m2*g*sin(t1+t2);

       B1=beta1*w1;
       B2=beta2*w2;

       C1=mu1;
       C2=mu2;
```

```
        /* Calculate torques at this time t */

        /*tq1=[[A(1,1:x(2)).*a1+A(1,x(2)+1:2*x(2)).*a2;A(2,1:x(2)).*a1+A(2,x↙
(2)+1:2*x(2)).*a2]+B+H+G]    */

        tq1=A11*a1+A12*a2+B1+H1+G1+C1;
        tq2=A21*a1+A22*a2+B2+H2+G2+C2;

        y[0]=t1;
        y[1]=w1;
        y[2]=a1;
        y[3]=tq1;
        y[4]=t2;
        y[5]=w2;
        y[6]=a2;
        y[7]=tq2;
        }

    else if( t <= (duration + deltat) && t > duration )
        { t=t-1;

        /* Generate position, velocity and acceleration at this time t */

        t1 =  th1ra + t * w1r;
        t2 =  th2ra + t * w2r;

        w1 = w1r;
        w2 = w2r;

        a1 = 0;
        a2 = 0;

        /* Calculate torque components */

        A11=I133+I233-.00273*m2*cos(t2)-.0260*m2*l1+.0110*m2+.105*cos(t2)*m2↙
*l1;
        A12=I233+.0110*m2-.00273*m2*cos(t2);
        A21=I233+.105*m2*l1*cos(t2)+.0110*m2;
        A22=I233+.0110*m2;

        H1=.0027*m2*sin(t2)*w2*w2+.00546*sin(t2)*w1*w2+(.00273*m2*sin(t2)+.1↙
05*m2*l1*sin(t2))*w1*w1;
        H2=.105*m2*l1*sin(t2)*w1*w1;

        G1=.105*m2*g*sin(t1+t2)-(.013*m1+.026*m2)*g*sin(t1);
        G2=.105*m2*g*sin(t1+t2);

        B1=beta1*w1;
        B2=beta2*w2;
```



```
        C1=mu1;
        C2=mu2;

        /* Calculate torques at this time t */

        /*tq1=[[A(1,1:x(2)).*a1+A(1,x(2)+1:2*x(2)).*a2;A(2,1:x(2)).*a1+A(2,x↙
(2)+1:2*x(2)).*a2]+B+H+G]    */

        tq1=A11*a1+A12*a2+B1+H1+G1+C1;
        tq2=A21*a1+A22*a2+B2+H2+G2+C2;

        y[0]=t1;
        y[1]=w1;
        y[2]=a1;
        y[3]=tq1;
        y[4]=t2;
        y[5]=w2;
        y[6]=a2;
        y[7]=tq2;

        }

    else if( t > (duration+deltat) && t <= (2*duration +deltat) )
        { t=t-1-deltat;

        /* Generate coefficients of cubic polynomial.*/

        th11 = th1rb;
        th12 = w1r;
        th13 =  (3 * (th1f-th1rb) - duration * (2 * w1r + w1f)) / (duration*↙
duration);
        th14 = (-2 * (th1f-th1rb) + duration * (w1r + w1f)) / (duration*dura↙
tion*duration);

        th21 = th2rb;
        th22 = w2r;
        th23 =  (3 * (th2f-th2rb) - duration * (2 * w2r + w2f)) / (duration*↙
duration);
        th24 = (-2 * (th2f-th2rb) + duration * (w2r + w2f)) / (duration*dura↙
tion*duration);

        t1 =  th11 + t * (th12 + t * (th13 + t * th14));
        t2 =  th21 + t * (th22 + t * (th23 + t * th24));

        w1 = th12+t*(2*th13+3*t*th14);
        w2 = th22+t*(2*th23+3*t*th24);

        a1 = 2*th13+6*th14*t;
```

```
        a2 = 2*th23+6*th24*t;

        A11=I133+I233-.00273*m2*cos(t2)-.0260*m2*l1+.0110*m2+.105*cos(t2)*m2↙
*l1;
        A12=I233+.0110*m2-.00273*m2*cos(t2);
        A21=I233+.105*m2*l1*cos(t2)+.0110*m2;
        A22=I233+.0110*m2;

        H1=.0027*m2*sin(t2)*w2*w2+.00546*sin(t2)*w1*w2+(.00273*m2*sin(t2)+.1↙
05*m2*l1*sin(t2))*w1*w1;
        H2=.105*m2*l1*sin(t2)*w1*w1;

        G1=.105*m2*g*sin(t1+t2)-(.013*m1+.026*m2)*g*sin(t1);
        G2=.105*m2*g*sin(t1+t2);

        B1=beta1*w1;
        B2=beta2*w2;

        C1=mu1;
        C2=mu2;

        /*tq1=[[A(1,1:x(2)).*a1+A(1,x(2)+1:2*x(2)).*a2;A(2,1:x(2)).*a1+A(2,x↙
(2)+1:2*x(2)).*a2]+B+H+G]    */

        tq1=A11*a1+A12*a2+B1+H1+G1+C1;
        tq2=A21*a1+A22*a2+B2+H2+G2+C2;

        y[0]=t1;
        y[1]=w1;
        y[2]=a1;
        y[3]=tq1;
        y[4]=t2;
        y[5]=w2;
        y[6]=a2;
        y[7]=tq2;
        }

    else
         {
         y[0]=0;
         y[1]=0;
         y[2]=0;
         y[3]=0;
         y[4]=0;
         y[5]=0;
         y[6]=0;
         y[7]=0;
         }
    }
```



```
}


/* Function: mdlTerminate =================================================↙
====
 * Abstract:
 *    In this function, you should perform any actions that are necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}


/*=====================================================*
 * See sfuntmpl_doc.c for the optional S-function methods *
 *=====================================================*/

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Appendix F Test Data

Acceleration Feedback	1000 Hz		Gains				Link Lengths (m)		Release Angles (radians)							
			Shoulder		Elbow		l1	0.3048	theta1	2.35619449						
		Position	161.7 kt		156.2 kt		l2	0.2936	theta2	1.57079633						
		Velocity	115.38	0.19	69.42	0.17					178 Data Points					
Actual		Accel.	1.232		0.621											
Ang Vel's																
Joint 1	4.7	7.05	9.45	7.05	9.45	11.8	11.8	11.8	14.15	14.15	14.15	16.5	16.5	16.5	18.85	
Joint 2	0.1	-4.65	-9.45	0.1	-4.65	-9.4	0.2	-4.6	-9.35	-4.55	0.25	-9.3	-4.5	0.8	-9.25	
	49	57	71	77	91	117	127	126	159	156	143	184	180	170	222	
	49	56	73	78	91	114	131	126	159	156	144	189	175	171	215	
	49	57	74	79	92	116	129	128	162	157	152	185	168	159	217	
	48	57	73	79	90	116	129	126	157	161	150	188	169	174	221	
	49	56	73	78	91	116	131	128	161	153	145	189	171	153	212	
	49	57	72	78	92	117	128	130	157	154	153	182	175	149	215	
	49	57	73	79	92	117	128	126	157	153	139	180	171	165	225	
	48	56	75	79	92	118	129	126	158	161	138	186	181	163	220	
									160	155	148	189	174	161	226	
									156	151	147	191	178	154	217	
									157	155	146	187	177	155	216	
									158	154	150	179	179	163	218	
											147	187	180	156	220	
											158	187	181	152	212	
											153	189	184	161	219	
											153	189	172	154	222	
											153	181	183	151	211	
											143	190	184		222	
															224	
Spread (Biggest -Smallest)	1	1	4	2	2	4	4	4	6	10	15	11	16	25	15	
Commanded																
X Velocity	-2.00948	-2.01772	-2.03672	-3.00385	-3.03323	-3.0414643	-5.03449	-4.0379757	-4.0462064	-5.0427179	-6.03923	-5.05095	-6.04746	-7.04397	-6.05569	
Y Velocity	0.016461	1.021204	2.036722	0.035072	1.040211	2.04495281	0.05193	1.048441367	2.05318353	1.05667209	0.060161	2.061414	1.064903	0.068391	2.069645	
Theoretical Distance	-53.7308	-62.7279	-73.4878	-80.5441	-94.5661	-109.87048	-135.336	-126.044883	-146.33924	-157.60063	-162.547	-182.894	-189.233	-189.825	-219.536	
Measured																
Average Distances	-47.4444	-54.4444	-69.3333	-73.8889	-85.4444	-107.88889	-119.111	-117.333333	-150.69231	-148.30769	-144.895	-181	-172.316	-156	-222.632	
Departure Angle	0.469349	26.84483	45	0.668947	18.92878	33.9152289	0.590976	14.55517797	26.9047957	11.8347624	0.570741	22.20156	9.986882	0.556279	18.86882	
																Averages
Error (desired-actual)	-6.28638	-8.28348	-4.1545	-6.65525	-9.12166	-1.981591	-16.2252	-8.71154935	4.35306963	-9.2929345	-17.6522	-1.89422	-16.9177	-33.8249	3.096057	-8.90349
Standard Deviation	0.46291	0.517549	1.195229	0.744024	0.744024	1.18773494	1.414214	1.511857892	1.83195541	3.03015151	5.345574	3.622731	5.128416	7.442426	4.42481	2.573574
Near Zero Departure	0.469349	0.668947	0.590976	0.570741	0.556279											
	11.69977	8.262865	11.98877	10.85976	17.81902											
	-6.28638	-6.65525	-16.2252	-17.6522	-33.8249											
	0.46291	0.744024	1.414214	5.345574	7.442426											
	-53.7308	-80.5441	-135.336	-162.547	-189.825											
10-45 Degrees	9.986882	11.83476	14.55518	18.92878	18.86882	22.2015589	26.84483	26.90479568	33.9152289	45						
	8.940105	5.896509	6.911466	9.645806	-1.41028	1.03568959	13.20541	-2.97464282	1.80357003	5.65331806						
	-16.9177	-9.29293	-8.71155	-9.12166	3.096057	-1.8942164	-8.28348	4.353069635	-1.981591	-4.154501						
	5.128416	3.030152	1.511858	0.744024	4.42481	3.62273143	0.517549	1.831955405	1.18773494	1.19522861						
	-189.233	-157.601	-126.045	-94.5661	-219.536	-182.89422	-62.7279	-146.339238	-109.87048	-73.487834						
20-50 Degrees	26.84483	45	33.91523	26.9048	22.20156											
	13.20541	5.653318	1.80357	-2.97464	1.03569											
	-8.28348	-4.1545	-1.98159	4.35307	-1.89422											
	0.517549	1.195229	1.187735	1.831955	3.622731											
	-62.7279	-73.4878	-109.87	-146.339	-182.894											

Model Based 1000 Hz

l1	0.3048	theta1	2.35619449
l2	0.2936	theta2	1.57079633

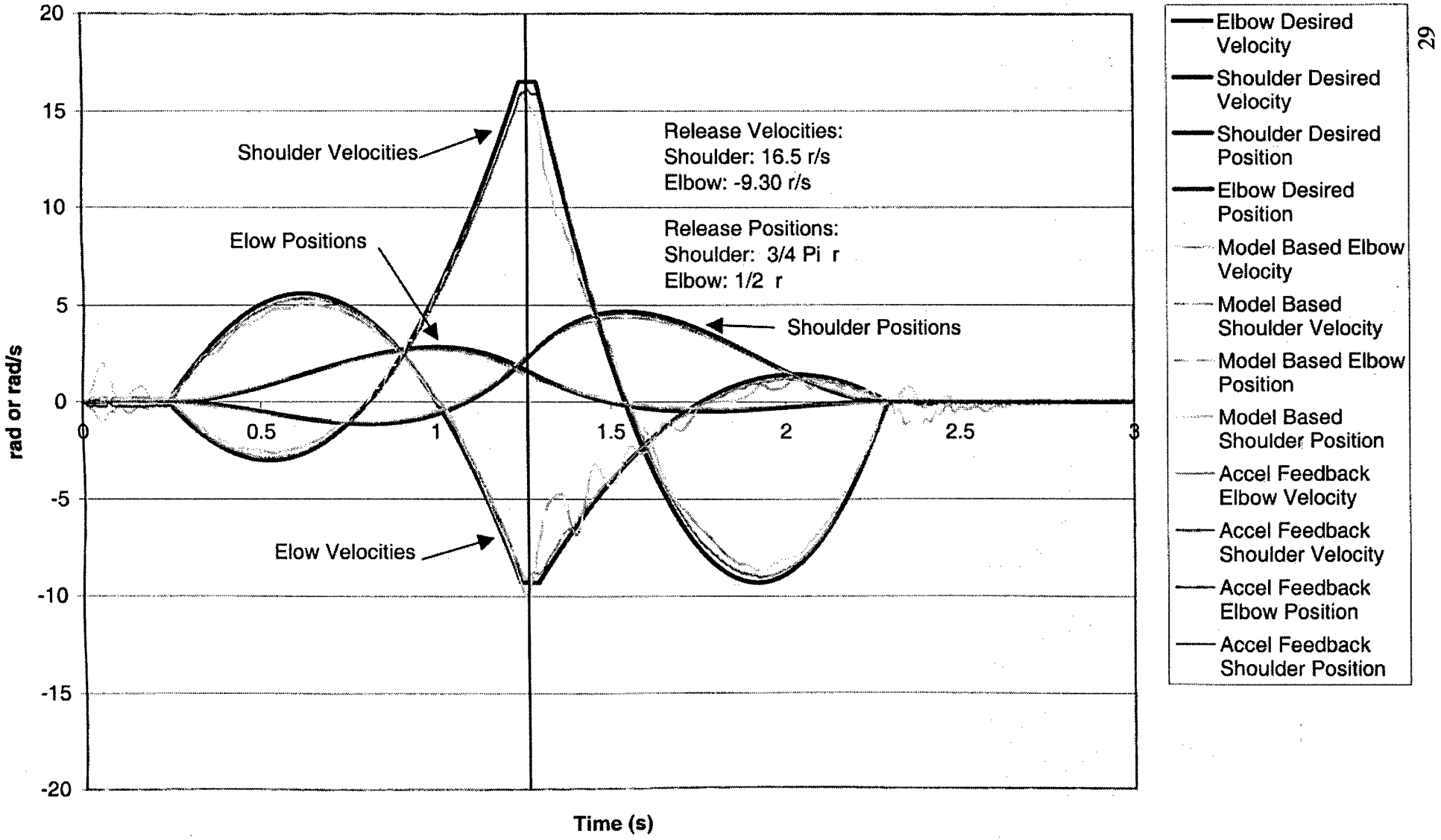
121 Data Points

Actual Ang Vel's

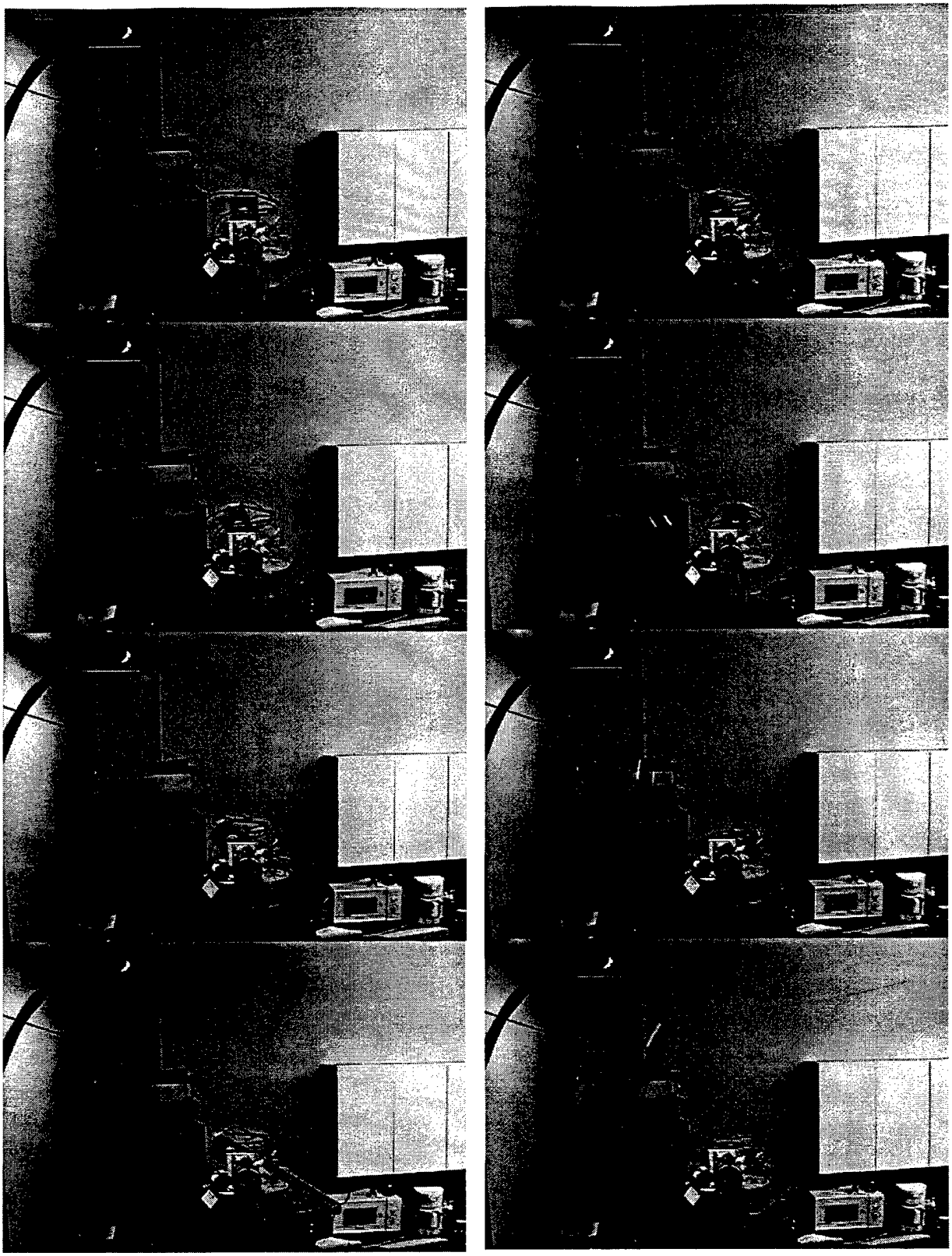
Joint 1	4.7	7.05	9.45	7.05	9.45	11.8	11.8	11.8	14.15	14.15	14.15	14.15	16.5	16.5	18.85
Joint 2	0.1	-4.65	-9.45	0.1	-4.65	-9.4	0.2	-4.6	-9.35	-4.55	0.25	-9.3	-4.5	-9.3	-9.25
	49	62	75	77	92	111	117	114	132	140	128	142	159	160	194
	48	62	74	79	92	109	121	118	141	132	131	141	144	170	195
	49	60	73	78	92	109	123	115	138	131	123	136	153	168	191
	49	62	74	80	93	107	116	116	141	135	125	140	153	171	190
	49	60	74	80	91	109	128	113	143	138	140	140	141	161	179
	49	62	74	81	92	109	124	116	134	132	126	133	144	172	184
							117	114	137	130	126	140	144	170	190
							128	115	140	127	134	140	154	161	191
							124	117	137	133	141	144	158	169	195
													159	167	178
															197
Spread	1	2	2	4	2	4	12	5	11	10	18	11	18	12	19
Commanded															
X	-2.009484	-2.017715	-2.036722	-3.003846	-3.033234	-3.0414643	-5.034487	-4.0379757	-4.0462064	-5.0427179	-6.039229	-4.056587	-5.899364	-5.050949	-6.055691
Y	0.016461	1.021204	2.036722	0.035072	1.040211	2.04495281	0.05193	1.048441367	2.05318353	1.05667209	0.060161	2.042803	1.062131	2.061414	2.069645
Range	-53.73083	-62.72792	-73.48783	-80.54414	-94.56611	-109.87048	-135.3363	-126.044883	-146.33924	-157.60063	-162.547	-146.4956	-184.5232	-182.8942	-219.5355
Measured															
Average Ranges	-46	-56.85714	-67.71429	-72.42857	-83.14286	-98	-115	-108.3	-129.4	-124.8	-123.2	-130.7	-142.8182	-156.8182	-179.25
Departure Angle	0.469349	26.84483	45	0.668947	18.92878	33.9152289	0.590976	14.55517797	26.9047957	11.8347624	0.570741	26.72877	10.20628	22.20156	18.86882
															Averages
Error (desired-actual)	-7.730825	-5.870781	-5.773549	-8.115572	-11.42325	-11.87048	-20.33626	-17.7448827	-16.939238	-32.800627	-39.34695	-15.79564	-41.70505	-26.07603	-40.28552	-20.12098
Standard Deviation	0.408248	1.032796	0.632456	1.47196	0.632456	1.26491106	4.582576	1.58113883	3.55121263	4.01386486	6.578585	3.244654	6.999206	4.532598	6.439932	3.131106
Near Zero Departure	0.469349	0.668947	0.590976	0.570741	22.20156										
	14.38806	10.07593	15.02647	24.20652	14.25744										
	-7.730825	-8.115572	-20.33626	-39.34695	-26.07603										
	0.408248	1.47196	4.582576	6.578585	4.532598										
	-53.73083	-80.54414	-135.3363	-162.547	-182.8942										
~10-23 Degrees	26.84483	18.92878	14.55518	26.9048	11.83476	26.7287675	10.20628	18.86881746							
	9.359119	12.07965	14.07823	11.57532	20.8125	10.7823248	22.60152	18.35034346							
	-5.870781	-11.42325	-17.74488	-16.93924	-32.80063	-15.795635	-41.70505	-40.2855224							
	1.032796	0.632456	1.581139	3.551213	4.013865	3.24465372	6.999206	6.439932241							
	-62.72792	-94.56611	-126.0449	-146.3392	-157.6006	-146.49564	-184.5232	-219.535522							
20-50 Degrees	26.84483	45	33.91523	26.9048	26.72877										
	9.359119	7.856469	10.80407	11.57532	10.78232										
	-5.870781	-5.773549	-11.87048	-16.93924	-15.79564										
	1.032796	0.632456	1.264911	3.551213	3.244654										
	-62.72792	-73.48783	-109.8705	-146.3392	-146.4956										

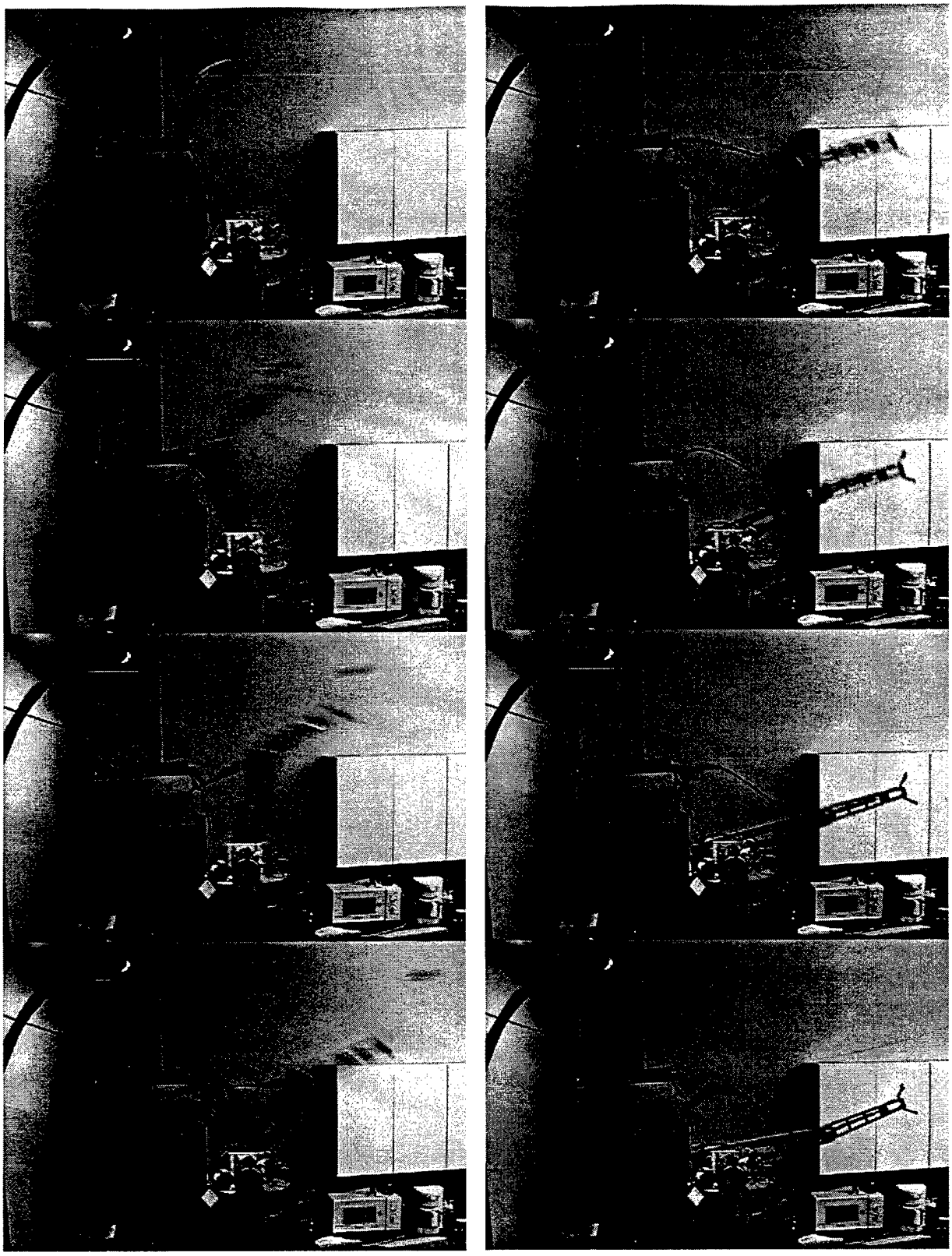
Appendix G Overlaid Response

Model Based Controller vs Acceleration Feedback Controller



Appendix H Video Capture Frames





Appendix I Agilent Data Sheets and Quadrature Decoder Circuit Diagram

Agilent Technologies
Innovating the HP Way

Quadrature Decoder/Counter Interface ICs

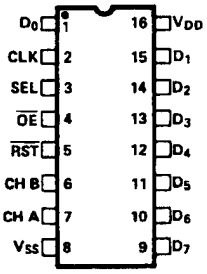
Technical Data

HCTL-2000
HCTL-2016
HCTL-2020

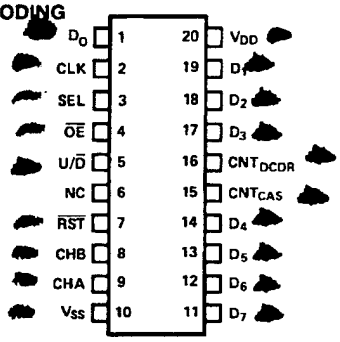
Features

- **Interfaces Encoder to Microprocessor**
- **14 MHz Clock Operation**
- **Full 4X Decode**
- **High Noise Immunity:** Schmitt Trigger Inputs Digital Noise Filter
- **12 or 16-Bit Binary Up/ Down Counter**
- **Latched Outputs**
- **8-Bit Tristate Interface**
- **8, 12, or 16-Bit Operating Modes**
- **Quadrature Decoder Output Signals, Up/Down and Count**
- **Cascade Output Signals, Up/ Down and Count**
- **Substantially Reduced System Software**

DIGITAL MOTION ENCODING



PINOUT A



PINOUT B

Applications

- **Interface Quadrature Incremental Encoders to Microprocessors**
- **Interface Digital Potentiometers to Digital Data Input Buses**

Description

The HCTL-2000, 2016, 2020 are CMOS ICs that perform the quadrature decoder, counter, and bus interface function. The HCTL-20XX family is designed to improve system performance

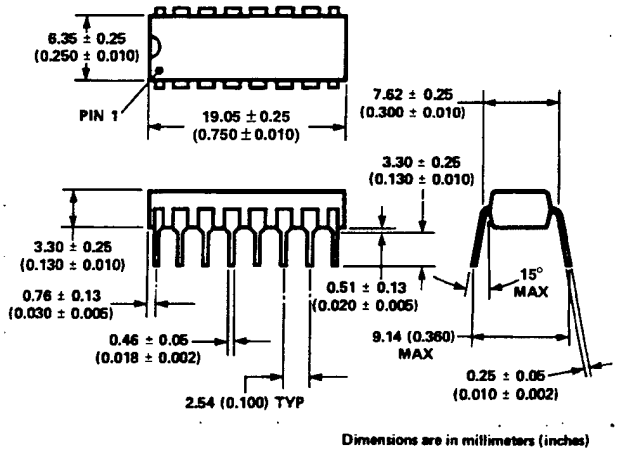
Devices

Part Number	Description	Package Drawing
HCTL-2000	12-bit counter. 14 MHz clock operation.	A
HCTL-2016	All features of the HCTL-2000. 16-bit counter.	A
HCTL-2020	All features of the HCTL-2016. Quadrature decoder output signals. Cascade output signals.	B

ESD WARNING: Standard CMOS handling precautions should be observed with the HCTL-20XX family ICs.

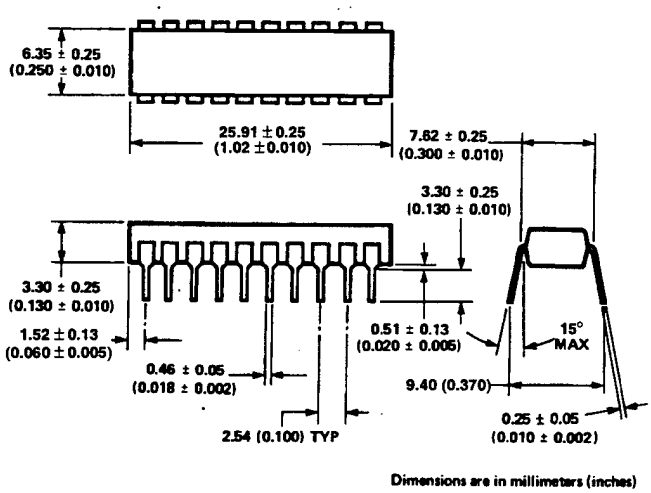
in digital closed loop motion control systems and digital data input systems. It does this by shifting time intensive quadrature decoder functions to a cost effective hardware solution. The entire HCTL-20XX family consists of a 4x quadrature decoder, a binary up/down state counter, and an 8-bit bus interface. The use of Schmitt-triggered CMOS inputs and input noise filters allows reliable operation in noisy environments. The HCTL-2000 contains a 12-bit counter. The HCTL-2016 and 2020 contain a 16-bit counter. The HCTL-2020 also contains quadrature decoder output signals and cascade signals for use with many standard counter ICs. The HCTL-20XX family provides LSTTL compatible tri-state output buffers. Operation is specified for a temperature range from -40 to +85°C at clock frequencies up to 14 MHz.

Package Dimensions



PACKAGE A LEAD FINISH: SOLDER DIPPED

PACKAGE A



PACKAGE B LEAD FINISH: SOLDER DIPPED

PACKAGE B

Operating Characteristics

Table 1. Absolute Maximum Ratings

(All voltages below are referenced to V_{SS})

Parameter	Symbol	Limits	Units
DC Supply Voltage	V_{DD}	-0.3 to +5.5	V
Input Voltage	V_{IN}	-0.3 to V_{DD} +0.3	V
Storage Temperature	T_S	-40 to +125	°C
Operating Temperature	T_A[1]	-40 to +85	°C

Table 2. Recommended Operating Conditions

Parameter	Symbol	Limits	Units
DC Supply Voltage	V_{DD}	+4.5 to +5.5	V
Ambient Temperature	T_A[1]	-40 to +85	°C

Table 3. DC Characteristics V_{DD} = 5 V ± 5%; T_A = -40 to 85°C

Symbol	Parameter	Condition	Min.	Typ.	Max.	Unit
V_{IL}[2]	Low-Level Input Voltage				1.5	V
V_{IH}[2]	High-Level Input Voltage		3.5			V
V_{T+}	Schmitt-Trigger Positive-Going Threshold			3.5	4.0	V
V_{T-}	Schmitt-Trigger Negative-Going Threshold		1.0	1.5		V
V_H	Schmitt-Trigger Hysteresis		1.0	2.0		V
I_{IN}	Input Current	$V_{IN} = V_{SS}$ or V_{DD}	-10	1	+10	µA
V_{OH}[2]	High-Level Output Voltage	I_{OH} -1.6 mA	2.4	4.5		V
V_{OL}[2]	Low-Level Output Voltage	I_{OL} = +4.8 mA		0.2	0.4	V
I_{OZ}	High-Z Output Leakage Current	$V_O = V_{SS}$ or V_{DD}	-10	1	+10	µA
I_{DD}	Quiescent Supply Current	$V_{IN} = V_{SS}$ or V_{DD}, V_O = HiZ		1	5	µA
C_{IN}	Input Capacitance	Any Input[3]		5		pF
C_{OUT}	Output Capacitance	Any Output[3]		6		pF

Notes:
1. Free air.
2. In general, for any V_{DD} between the allowable limits (+4.5 V to +5.5 V), V_{IL} = 0.3 V_{DD} and V_{IH} = 0.7 V_{DD}; typical values are $V_{OH} = V_{DD}$ - 0.5 V @ I_{OH} = -40 µA and $V_{OL} = V_{SS}$ + 0.2 V @ I_{OL} = 1.6 mA.
3. Including package capacitance.

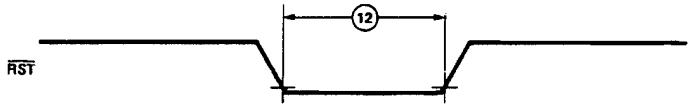


Figure 1. Reset Waveform.

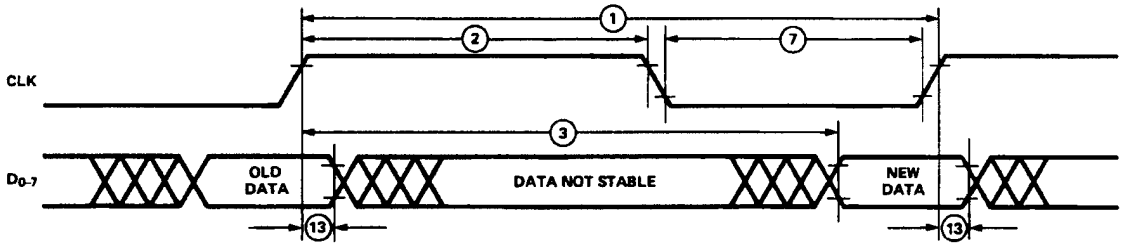


Figure 2. Waveform for Positive Clock Related Delays.

Functional Pin Description

Table 4. Functional Pin Descriptions

<table>
<tr><th>Symbol</th><th>Pin 2000/2016</th><th>Pin 2020</th><th>Description</th></tr>
<tr><td>V_{DD}</td><td>16</td><td>20</td><td>Power Supply</td></tr>
<tr><td>V_{SS}</td><td>8</td><td>10</td><td>Ground</td></tr>
<tr><td>CLK</td><td>2</td><td>2</td><td>CLK is a Schmitt-trigger input for the external clock signal.</td></tr>
<tr><td>CHA
CHB</td><td>7
6</td><td>9
8</td><td>CHA and CHB are Schmitt-trigger inputs which accept the outputs from a quadrature encoded source, such as incremental optical shaft encoder. Two channels, A and B, nominally 90 degrees out of phase, are required.</td></tr>
<tr><td>$\overline{RST}$</td><td>5</td><td>7</td><td>This active low Schmitt-trigger input clears the internal position counter and the position latch. It also resets the inhibit logic. $\overline{RST}$ is asynchronous with respect to any other input signals.</td></tr>
<tr><td>$\overline{OE}$</td><td>4</td><td>4</td><td>This CMOS active low input enables the tri-state output buffers. The $\overline{OE}$ and SEL inputs are sampled by the internal inhibit logic on the falling edge of the clock to control the loading of the internal position data latch.</td></tr>
<tr><td>SEL</td><td>3</td><td>3</td><td>This CMOS input directly controls which data byte from the position latch is enabled into the 8-bit tri-state output buffer. As in $\overline{OE}$ above, SEL also controls the internal inhibit logic.
<table>
<tr><th>SEL</th><th>BYTE SELECTED</th></tr>
<tr><td>0</td><td>High</td></tr>
<tr><td>1</td><td>Low</td></tr>
</table></td></tr>
<tr><td>CNT_{DCDR}</td><td></td><td>16</td><td>A pulse is presented on this LSTTL-compatible output when the quadrature decoder has detected a state transition.</td></tr>
<tr><td>U/$\overline{D}$</td><td></td><td>5</td><td>This LSTTL-compatible output allows the user to determine whether the IC is counting up or down and is intended to be used with the CNT_{DCDR} and CNT_{CAS} outputs. The proper signal U (high level) or $\overline{D}$ (low level) will be present before the rising edge of the CNT_{DCDR} and CNT_{CAS} outputs.</td></tr>
<tr><td>CNT_{CAS}</td><td></td><td>15</td><td>A pulse is presented on this LSTTL-compatible output when the HCTL-2020 internal counter overflows or underflows. The rising edge on this waveform may be used to trigger an external counter.</td></tr>
<tr><td>D0</td><td>1</td><td>1</td><td rowspan="8">These LSTTL-compatible tri-state outputs form an 8-bit output port through which the contents of the 12/16-bit position latch may be read in 2 sequential bytes. The high byte, containing bits 8-15, is read first (on the HCTL-2000, the most significant 4 bits of this byte are set to 0 internally). The lower byte, bits 0-7, is read second.</td></tr>
<tr><td>D1</td><td>15</td><td>19</td></tr>
<tr><td>D2</td><td>14</td><td>18</td></tr>
<tr><td>D3</td><td>13</td><td>17</td></tr>
<tr><td>D4</td><td>12</td><td>14</td></tr>
<tr><td>D5</td><td>11</td><td>13</td></tr>
<tr><td>D6</td><td>10</td><td>12</td></tr>
<tr><td>D7</td><td>9</td><td>11</td></tr>
<tr><td>NC</td><td></td><td>6</td><td>Not connected - this pin should be left floating.</td></tr>
</table>

Switching Characteristics

Table 5. Switching Characteristics Min/Max specifications at V_{DD} = 5.0 ± 5%, T_A = -40 to + 85°C.

Symbol Description			Min.	Max.	Units
1	t_{CLK}	Clock period	70		ns
2	t_{CHH}	Pulse width, clock high	28		ns
3	t_{CD}[1]	Delay time, rising edge of clock to valid, updated count information on D0-7		65	ns
4	t_{ODE}	Delay time, $\overline{OE}$ fall to valid data		65	ns
5	t_{ODZ}	Delay time, $\overline{OE}$ rise to Hi-Z state on D0-7		40	ns
6	t_{SDV}	Delay time, SEL valid to stable, selected data byte (delay to High Byte = delay to Low Byte)		65	ns
7	t_{CLH}	Pulse width, clock low	28		ns
8	t_{SS}[2]	Setup time, SEL before clock fall	20		ns
9	t_{OS}[2]	Setup time, $\overline{OE}$ before clock fall	20		ns
10	t_{SH}[2]	Hold time, SEL after clock fall	0		ns
11	t_{OH}[2]	Hold time, $\overline{OE}$ after clock fall	0		ns
12	t_{RST}	Pulse width, $\overline{RST}$ low	28		ns
13	t_{DCD}	Hold time, last position count stable on D0-7 after clock rise	10		ns
14	t_{DSD}	Hold time, last data byte stable after next SEL state change	5		ns
15	t_{DOD}	Hold time, data byte stable after $\overline{OE}$ rise	5		ns
16	t_{UDD}	Delay time, U/$\overline{D}$ valid after clock rise		45	ns
17	t_{CHD}	Delay time, CNT_{DCDR} or CNT_{CAS} high after clock rise		45	ns
18	t_{CLD}	Delay time, CNT_{DCDR} or CNT_{CAS} low after clock fall		45	ns
19	t_{UDH}	Hold time, U/$\overline{D}$ stable after clock rise	10		ns
20	t_{UDCS}	Setup time, U/$\overline{D}$ valid before CNT_{DCDR} or CNT_{CAS} rise	t_{CLK}-45		ns
21	t_{UDCH}	Hold time, U/$\overline{D}$ stable after CNT_{DCDR} or CNT_{CAS} rise	t_{CLK}-45		ns

Notes:

1. t_{CD} specification and waveform assume latch not inhibited.
2. t_{SS}, t_{OS}, t_{SH}, t_{OH} only pertain to proper operation of the inhibit logic. In other cases, such as 8 bit read operations, these setup and hold times do not need to be observed.

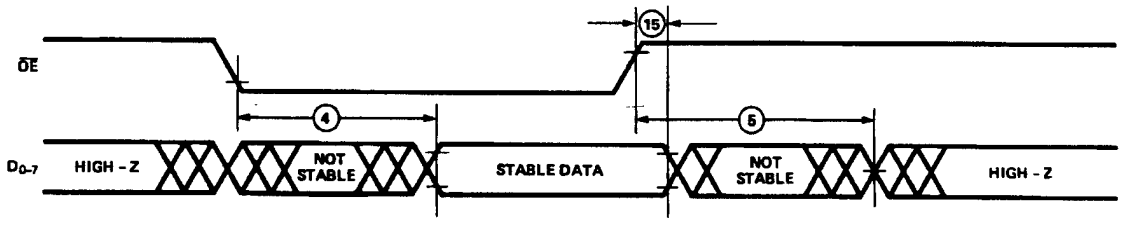


Figure 3. Tri-State Output Timing.

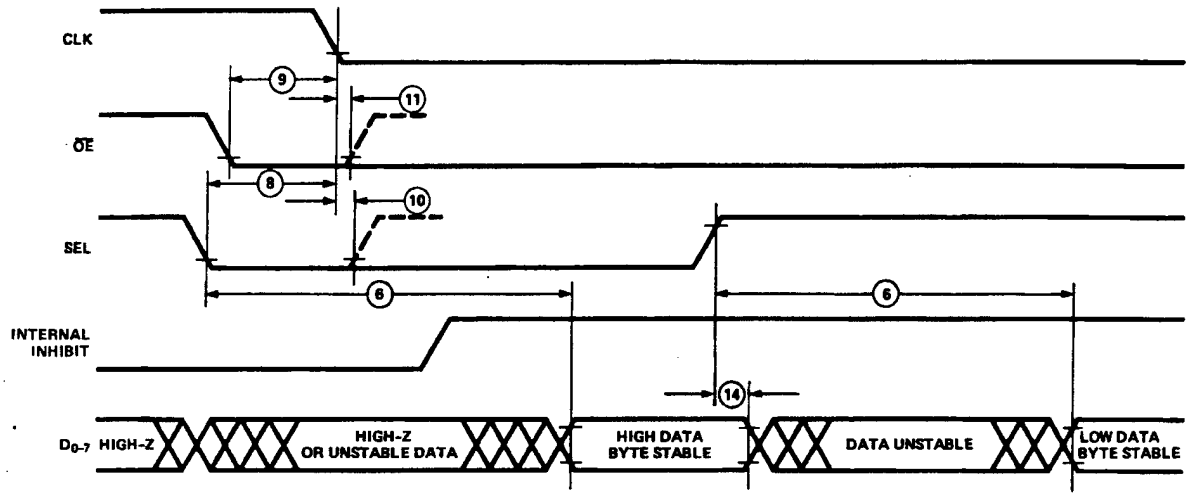


Figure 4. Bus Control Timing.

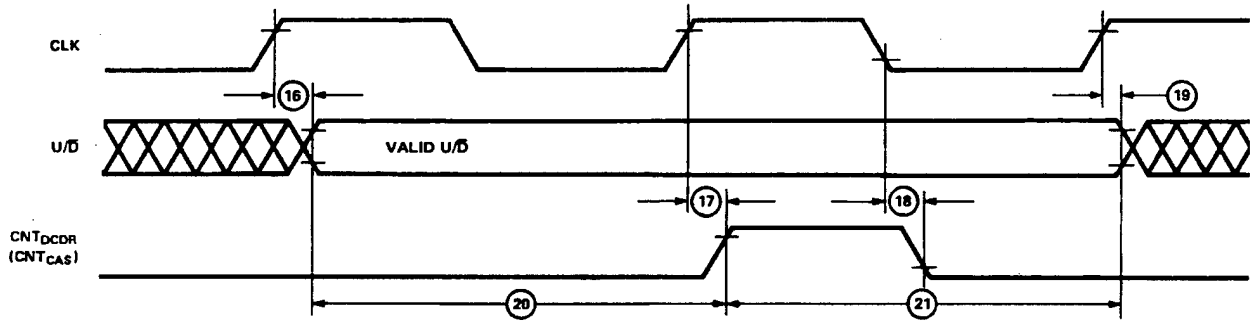


Figure 5. Decoder, Cascade Output Timing (HCTL-2020 only).

Operation

A block diagram of the HCTL-20XX family is shown in Figure 6. The operation of each major function is described in the following sections.

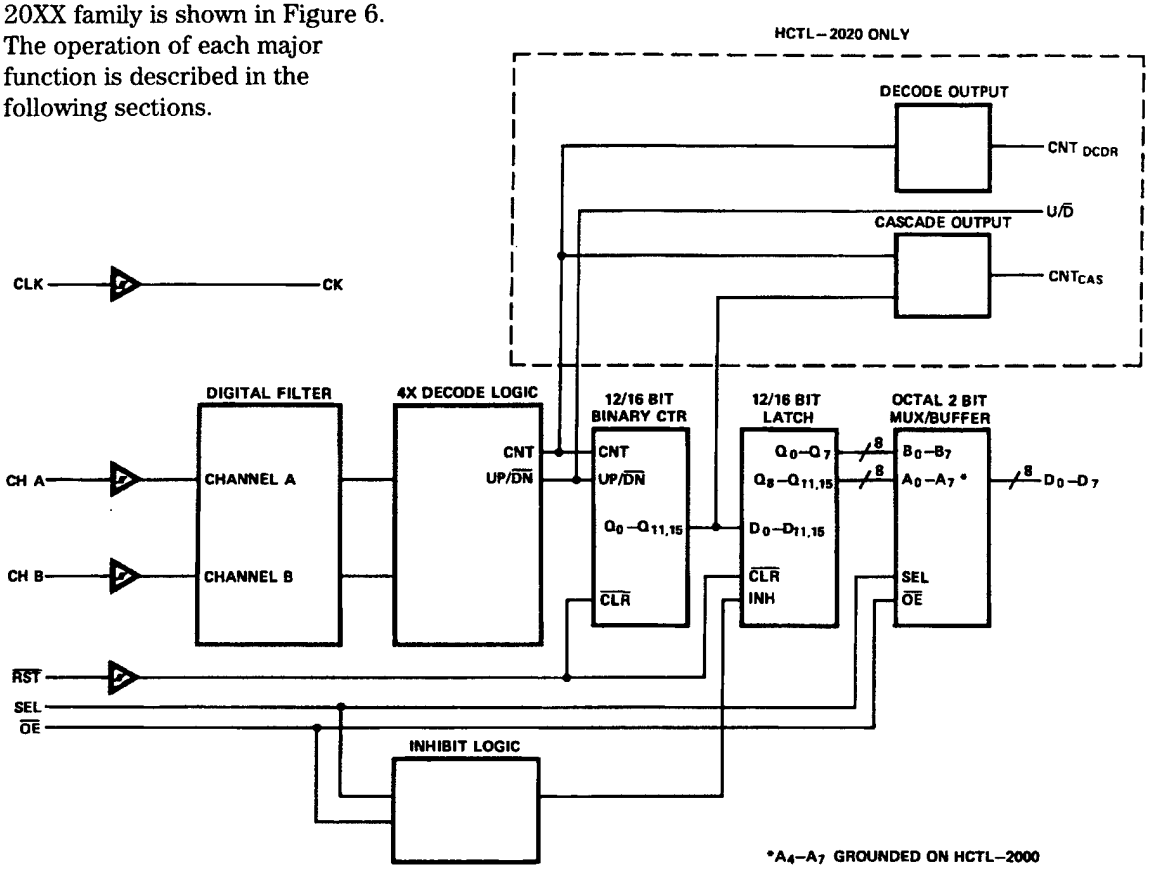


Figure 6. Simplified Logic Diagram.

Digital Noise Filter

The digital noise filter section is responsible for rejecting noise on the incoming quadrature signals. The input section uses two techniques to implement improved noise rejection. Schmitt-trigger inputs and a three-clock-cycle delay filter combine to reject low level noise and large, short duration noise spikes that typically occur in motor system applications. Both common mode and differential mode noise are rejected. The user benefits from these techniques by improved integrity of the data in the counter. False counts triggered by noise are avoided.

Figure 7 shows the simplified schematic of the input section. The signals are first passed through a Schmitt trigger buffer to address the problem of input signals with slow rise times and low level noise (approximately < 1 V). The cleaned up signals are then passed to a four-bit delay filter. The signals on each channel are sampled on rising clock edges. A time history of the signals is stored in the four-bit shift register. Any change on the input is tested for a stable level being present for three consecutive rising clock edges. Therefore, the filtered output waveforms can change only after an input level has the same value for three consecutive rising clock edges. Refer to Figure 8 which shows the timing diagram. The result of this circuitry is that short noise spikes between rising clock edges are ignored and pulses shorter than two clock periods are rejected.

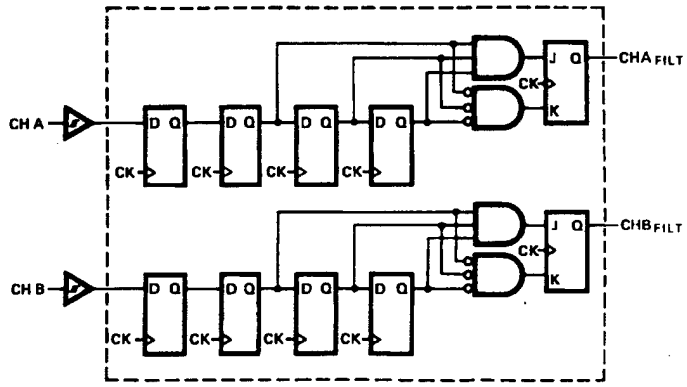


Figure 7. Simplified Digital Noise Filter Logic.

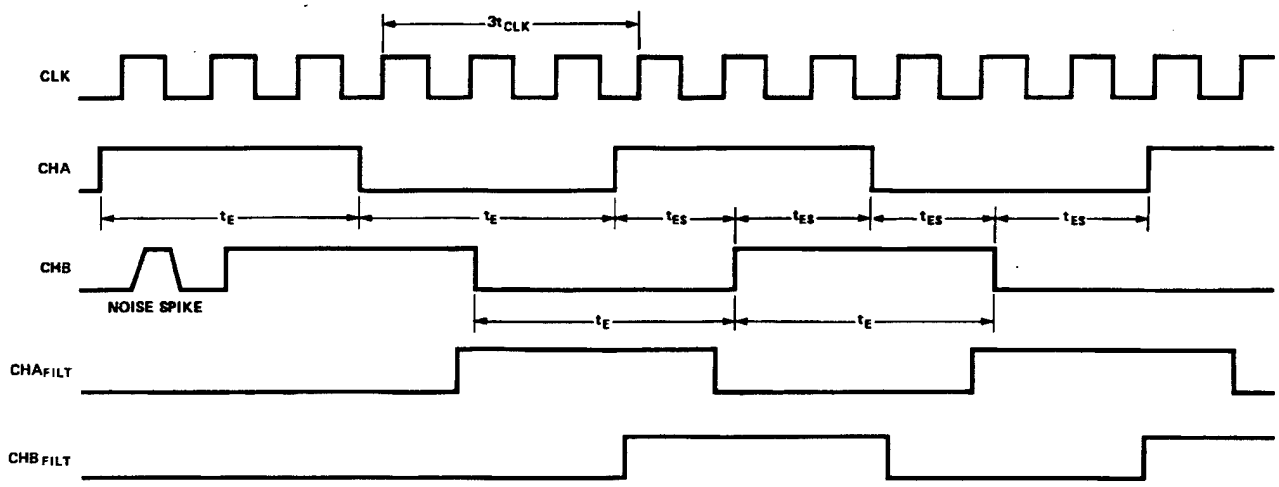


Figure 8. Signal Propagation through Digital Noise Filter.

Quadrature Decoder

The quadrature decoder decodes the incoming filtered signals into count information. This circuitry multiplies the resolution of the input signals by a factor of four (4X decoding). When using an encoder for motion sensing, the user benefits from the increased resolution by being able to provide better system control.

The quadrature decoder samples the outputs of the CHA and CHB filters. Based on the past binary state of the two signals and the present state, it outputs a count signal and a direction signal to the internal position counter. In the case of the HCTL-2020, the signals also go to external pins 5 and 16 respectively.

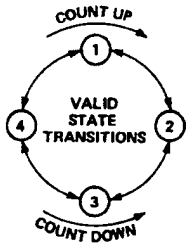
Figure 9 shows the quadrature states and the valid state transitions. Channel A leading channel B results in counting up. Channel B leading channel A results in counting down. Illegal state transitions, caused by faulty encoders or noise severe enough to pass through the filter, will produce an erroneous count.

Design Considerations

The designer should be aware that the operation of the digital filter places a timing constraint on the relationship between incoming quadrature signals and the external clock. Figure 8 shows the timing waveform with an incremental encoder input. Since an input has to be stable for three rising clock edges, the encoder pulse width (t_E - low or high) has to be greater than three clock periods ($3t_{CLK}$). This guarantees that the asynchronous input will be stable during three consecutive rising clock edges. A realistic design also has to take

into account finite rise times of the waveforms, asymmetry of the waveforms, and noise. In the presence of large amounts of noise, t_E should be much greater than $3t_{CLK}$ to allow for the interruption of the consecutive level sampling by the three-bit delay filter. It should be noted that a change on the inputs that is qualified by the filter will internally propagate in a maximum of seven clock periods.

The quadrature decoder circuitry imposes a second timing constraint between the external clock and the input signals. There must be at least one clock period between consecutive quadrature states. As shown in Figure 9, a quadrature state is defined by consecutive edges on both channels. Therefore, t_{ES} (encoder state period) > t_{CLK}. The designer must account for deviations from the nominal 90 degree phasing of input signals to guarantee that $t_{ES} > t_{CLK}$.



CHA	CHB	STATE
1	0	1
1	1	2
0	1	3
0	0	4

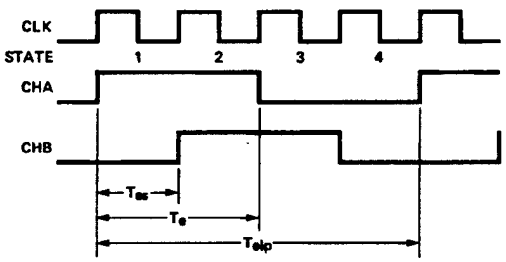


Figure 9. 4x Quadrature Decoding.

Position Counter

This section consists of a 12-bit (HCTL-2000) or 16-bit (HCTL-2016/2020) binary up/down counter which counts on rising clock edges as explained in the Quadrature Decoder Section. All 12 or 16 bits of data are passed to the position data latch. The system can use this count data in several ways:

A. System total range is ≤ 12 or 16 bits, so the count represents "absolute" position.

B. The system is cyclic with ≤ 12 or 16 bits of count per cycle. $\overline{RST}$ is used to reset the counter every cycle and the system uses the data to interpolate within the cycle.

C. System count is > 8, 12, or 16 bits, so the count data is used as a relative or incremental position input for a system software computation of absolute position. In this case counter rollover occurs. In order to prevent loss of position information, the processor must read the outputs of the IC before the count increments one-half of the maximum count capability (i.e. 127. 2047, or 32,767 quadrature counts). Two's-complement arithmetic is normally used to compute position from these periodic position updates. Three modes can be used:

1. The IC can be put in 8-bit mode by tying the SEL line high, thus simplifying IC interface. The outputs must then be read at least once every 127 quadrature counts.
2. The HCTL-2000 can be used in 12-bit mode and sampled at least once every 2047 quadrature counts.
3. The HCTL-2016 or 2020 can be used in 16-bit mode and sampled at least once every 32,767 quadrature counts.

D. The system count is > 16 bits so the HCTL-2020 can be cascaded with other standard counter ICs to give absolute position.

Position Data Latch

The position data latch is a 12/16-bit latch which captures the position counter output data on each rising clock edge, except when its inputs are disabled by the inhibit logic section during two-byte read operations. The output data is passed to the bus interface section. When active, a signal from the inhibit logic section prevents new data from being captured by the latch, keeping the data stable while successive reads are made through the bus section. The latch is automatically reenabled at the end of these reads. The latch is cleared to 0 asynchronously by the $\overline{RST}$ signal.

Inhibit Logic

The Inhibit Logic Section samples the OE and $\overline{SEL}$ signals on the falling edge of the clock and, in response to certain conditions (see Figure 10 below), inhibits the position data latch. The RST signal asynchronously clears the inhibit logic, enabling the latch. A simplified logic diagram of the inhibit circuitry is illustrated in Figure 11.

Bus Interface

The bus interface section consists of a 16 to 8 line multiplexer and an 8-bit, three-state output buffer. The multiplexer allows independent access to the low and high bytes of the position data latch. The SEL and $\overline{OE}$ signals determine which byte is output and whether or not the output bus is in the high-Z state. In the case of the HCTL-2000 the data latch is only 12 bits wide and the upper four bits of the high byte are internally set to zero.

Quadrature Decoder Output (HCTL-2020 Only)

The quadrature decoder output section consists of count and up/down outputs derived from the 4X decode logic of the HCTL-2020. When the decoder has detected a count, a pulse, one-half clock cycle long, will be output on the CNT_{DCDR} pin. This output will occur during the clock cycle in which the internal counter is updated. The U/D pin will be set to the proper voltage level one clock cycle before the rising edge of the CNT_{DCDR} pulse, and held one clock cycle after the rising edge of the CNT_{DCDR} pulse. These outputs are not affected by the inhibit logic. See Figures 5 and 12 for detailed timing.

Cascade Output (HCTL-2020 Only)

The cascade output also consists of count and up/down outputs. When the HCTL-2020 internal counter overflows or underflows, a pulse, one-half clock cycle long, will be output on the CNT_{CAS} pin. This output will occur during the clock cycle in which the internal counter is updated. The $U/\overline{D}$ pin will be set to the proper voltage level one clock cycle before the rising edge of the CNT_{CAS} pulse, and held one clock cycle after the rising edge of the CNT_{CAS} pulse. These outputs are not affected by the inhibit logic. See Figures 5 and 12 for detailed timing.

Step	SEL	$\overline{OE}$	CLK	Inhibit Signal	Action
1	L	L	⌐	1	Set inhibit; read high byte
2	H	L	⌐	1	Read low byte; starts reset
3	X	H	⌐	0	Completes inhibit logic reset

Figure 10. Two Byte Read Sequence.

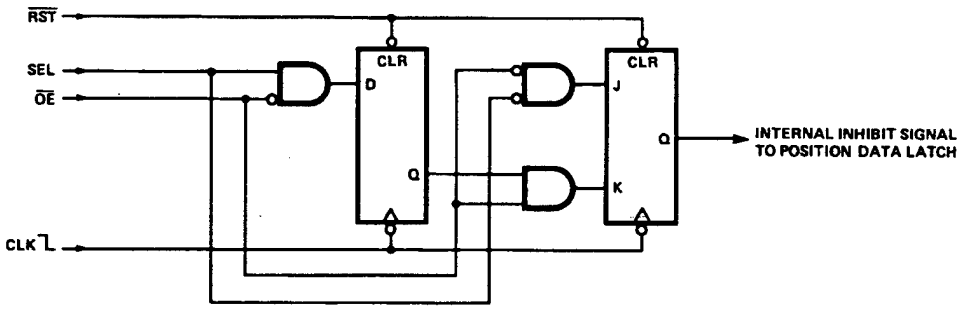
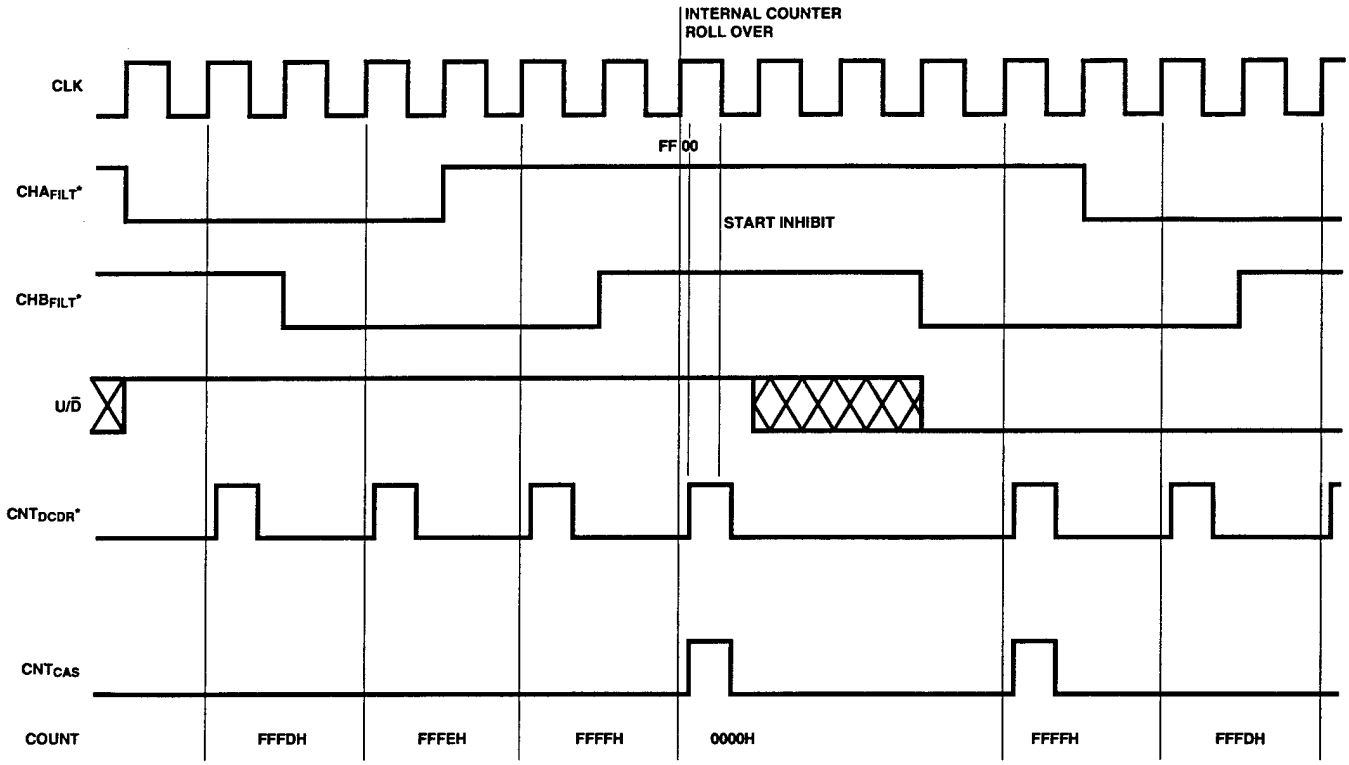


Figure 11. Simplified Inhibit Logic.



*CHA$_{FILT}$ AND CHB$_{FILT}$ ARE THE OUTPUTS OF THE DIGITAL NOISE FILTER (SEE FIGURES 7 AND 8).

Figure 12. Decode and Cascade Output Diagram.

Cascade Considerations (HCTL-2020 Only)

The HCTL-2020's cascading system allows for position reads of more than two bytes. These reads can be accomplished by latching all of the bytes and then reading the bytes sequentially over the 8-bit bus. It is assumed here that, externally, a counter followed by a latch is used to count any count that exceeds 16 bits. This configuration is compatible with the HCTL-2020 internal counter/latch combination.

Consider the sequence of events for a read cycle that starts as the HCTL-2020's internal counter rolls over. On the rising clock edge, count data is updated in the internal counter, rolling it over. A count-cascade pulse (CNT_{CAS}) will be generated with some delay after the rising clock edge (t_{CHD}). There will be additional propagation delays through the external counters and registers. Meanwhile, with SEL and OE low to start the read, the internal latches are inhibited at the falling edge and do not update again till the inhibit is reset. If the CNTCAS pulse now toggles the external counter and this count gets latched a major count error will occur. The count error is because the external latches get updated when the internal latch is inhibited.

Valid data can be ensured by latching the external counter data when the high byte read is started (SEL and OE low). This latched external byte corresponds to the count in the inhibited internal latch. The cascade pulse that occurs during the clock cycle when the read begins gets counted by the external counter and is not lost.

For example, suppose the HCTL-2020 count is at FFFFH and an external counter is at F0H, with the count going up. A count occurring in the HCTL-2020 will cause the counter to roll over and a cascade pulse will be generated. A read starting on this clock cycle will show FFFFH from the HCTL-2020. The external latch should read F0H, but if the host latches the count after the cascade signal propagates through, the external latch will read F1H.

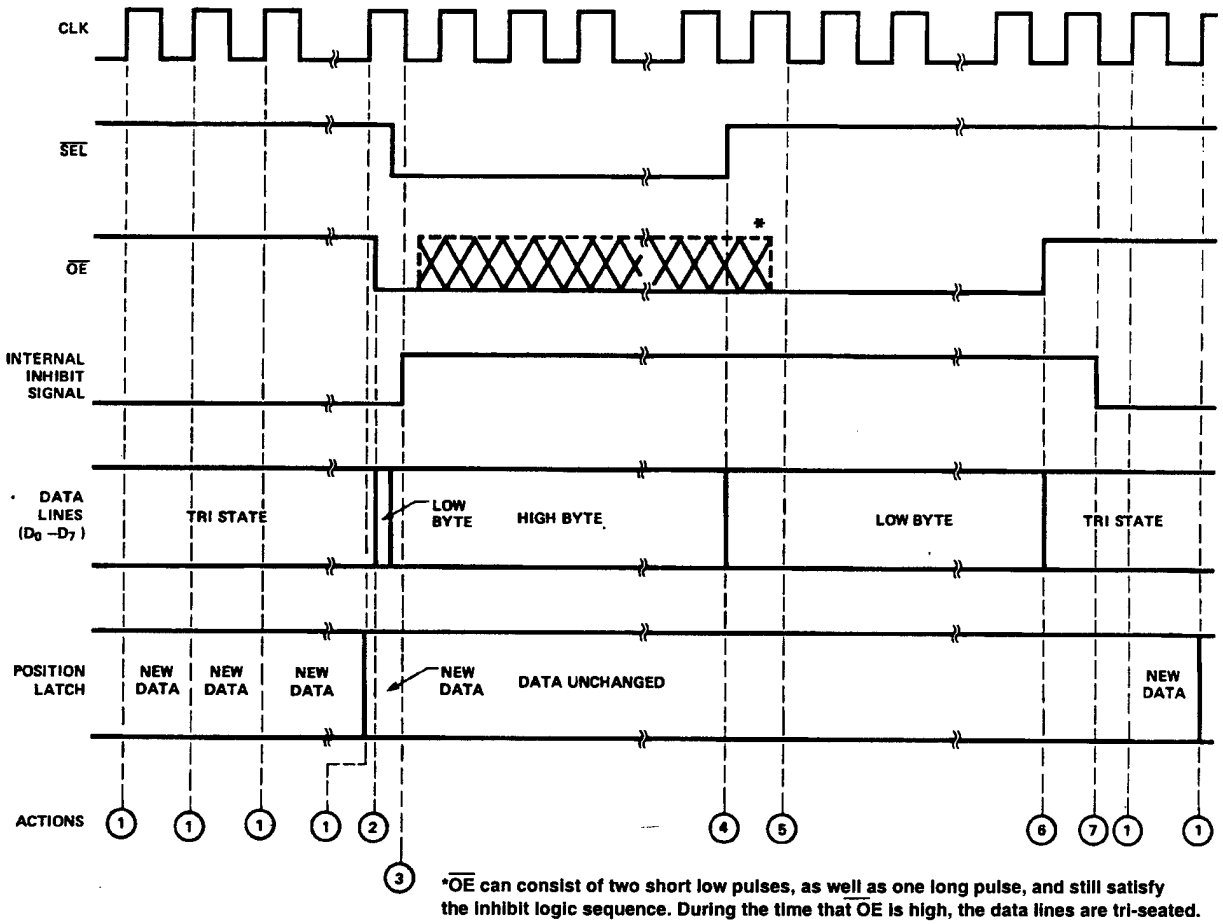
General Interfacing

The 12-bit (HCTL-2000) or 16-bit (HCTL-2016/2020) latch and inhibit logic allows access to 12 or 16 bits of count with an 8-bit bus. When only 8-bits of count are required, a simple 8-bit (1-byte) mode is available by holding SEL high continuously. This disables the inhibit logic. $\overline{OE}$ provides control of the tri-state bus, and read timing is shown in Figures 2 and 3.

For proper operation of the inhibit logic during a two-byte read, $\overline{OE}$ and SEL must be synchronous with CLK due to the falling edge sampling of $\overline{OE}$ and SEL.

The internal inhibit logic on the HCTL-20XX family inhibits the transfer of data from the counter to the position data latch during the time that the latch outputs are being read. The inhibit logic allows the microprocessor to first read the high order 4 or 8 bits from the latch and then read the low order 8 bits from the latch. Meanwhile, the counter can continue to keep track of the quadrature states from the CHA and CHB input signals.

Figure 11 shows the simplified inhibit logic circuit. The operation of the circuitry is illustrated in the read timing shown in Figure 13.



*$\overline{OE}$ can consist of two short low pulses, as well as one long pulse, and still satisfy the inhibit logic sequence. During the time that $\overline{OE}$ is high, the data lines are tri-seated.

Figure 13. Typical Interface Timing.

Actions

1. On the rising edge of the clock, counter data is transferred to the position data latch, provided the inhibit signal is low.
2. When $\overline{OE}$ goes low, the outputs of the multiplexer are enabled onto the data lines. If SEL is low, then the high order data bytes are enabled onto the data lines. If SEL is high, then the low order data bytes are enabled onto the data lines.
3. When the IC detects a low on $\overline{OE}$ and SEL during a falling clock edge, the internal inhibit signal is activated. This blocks new data from being transferred from the counter to the position data latch.
4. When SEL goes high, the data outputs change from the high byte to the low byte.
5. The first of two reset conditions for the inhibit logic is met when the IC detects a logic high on SEL and a logic low on $\overline{OE}$ during a falling clock edge.
6. When $\overline{OE}$ goes high, the data lines change to a high impedance state.
7. The IC detects a logic high on $\overline{OE}$ during a falling clock edge. This satisfies the second reset condition for the inhibit logic.

Interfacing the HCTL-2020 to a Motorola 6802/8 and Cascading the Counter for 24 Bits

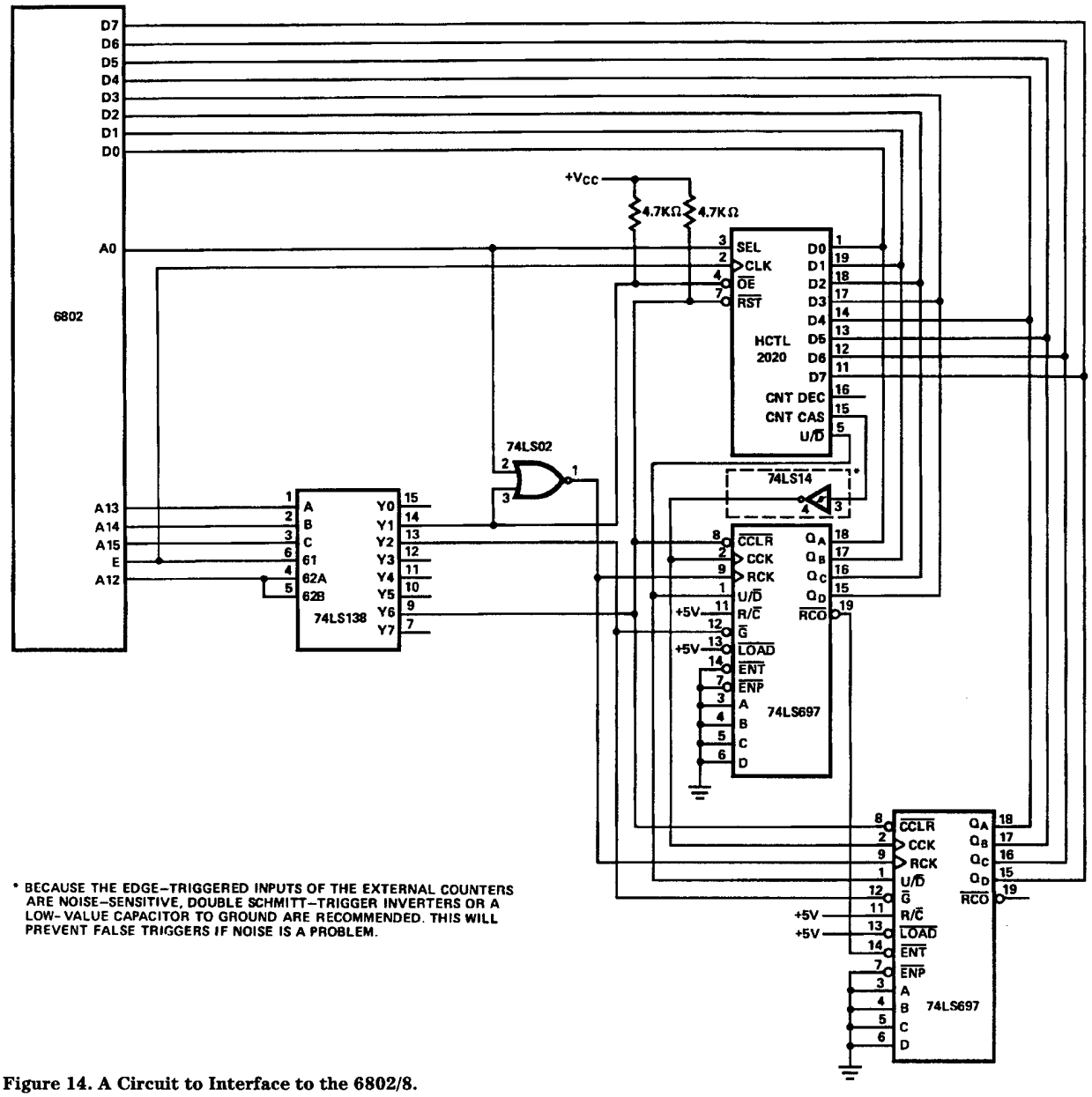


Figure 14. A Circuit to Interface to the 6802/8.

In this circuit an interface to a Motorola 6802/8 and a cascading scheme for a 24-bit counter are shown. This circuit provides a minimum part count by: 1) using two 74LS697 Up/Down counters with output registers and tri-state outputs and 2) using a Motorola 6802/8 LDX instruction which stores 16 bits of data into the index registers in two consecutive clock cycles.

The HCTL-2020 $\overline{OE}$ and the 74LS697 $\overline{G}$ lines are decoded from Address lines A15-A13. This results in counter data being enabled onto the bus whenever an external memory access is made to locations 4XXX or 2XXX. Address line A12 and processor clock E enables the 74LS138. The processor clock E is also used to clock the HCTL-2020. Address AO is connected directly to the SEL pin on the HCTL-2020. This line selects the low or high byte of data from the HCTL-2020.

Cascading is accomplished by connecting the CNT_{CAS} output on the HCTL-2020 with the counter clock (CCK) input on both 74LS697s. The $\overline{U/D}$ pin on the HCTL-2020 and the $\overline{U/D}$ pin on both 74LS697s are also directly connected for easy expansion. The $\overline{RCO}$ of the first 4-bit 74LS697 is connected to the $\overline{ENT}$ pin of the second 74LS697. This enables the second counter only when there is a $\overline{RCO}$ signal on the first counter.

This configuration allows the 6802 to read both data bytes with a single double-byte fetch instruction (LDX 2XX0). This instruction is a five cycle instruction which reads external memory location 2XX0 and stores the high order byte into the high byte of the index register. Memory location 2XX1 is next read and stored in the low order byte of the index register. The high byte of counter data is clocked into the 74LS697 registers when SEL is low and $\overline{OE}$ goes low. This upper byte can be read at any time by pulling the 74LS697 $\overline{G}$ low when reading address 4XXX. Figure 15 shows memory addresses and gives an example of reading the HCTL-2020. Figure 16 shows the interface timing for the circuit.

Address	Function
CXXX	Reset Counters
4XXX	Enable High Byte on Data Lines
2XX0	Enable Mid Byte on Data Lines
2XX1	Enable Low Byte on Data Lines

Read Example	
LDX 2000 STX 0100	Loads mid byte and then low byte into memory locations 0100 and 0101
LDAA 4000 STAA 0102	Loads the high byte into memory location 0102

Figure 15. Memory Addresses and Read Example.

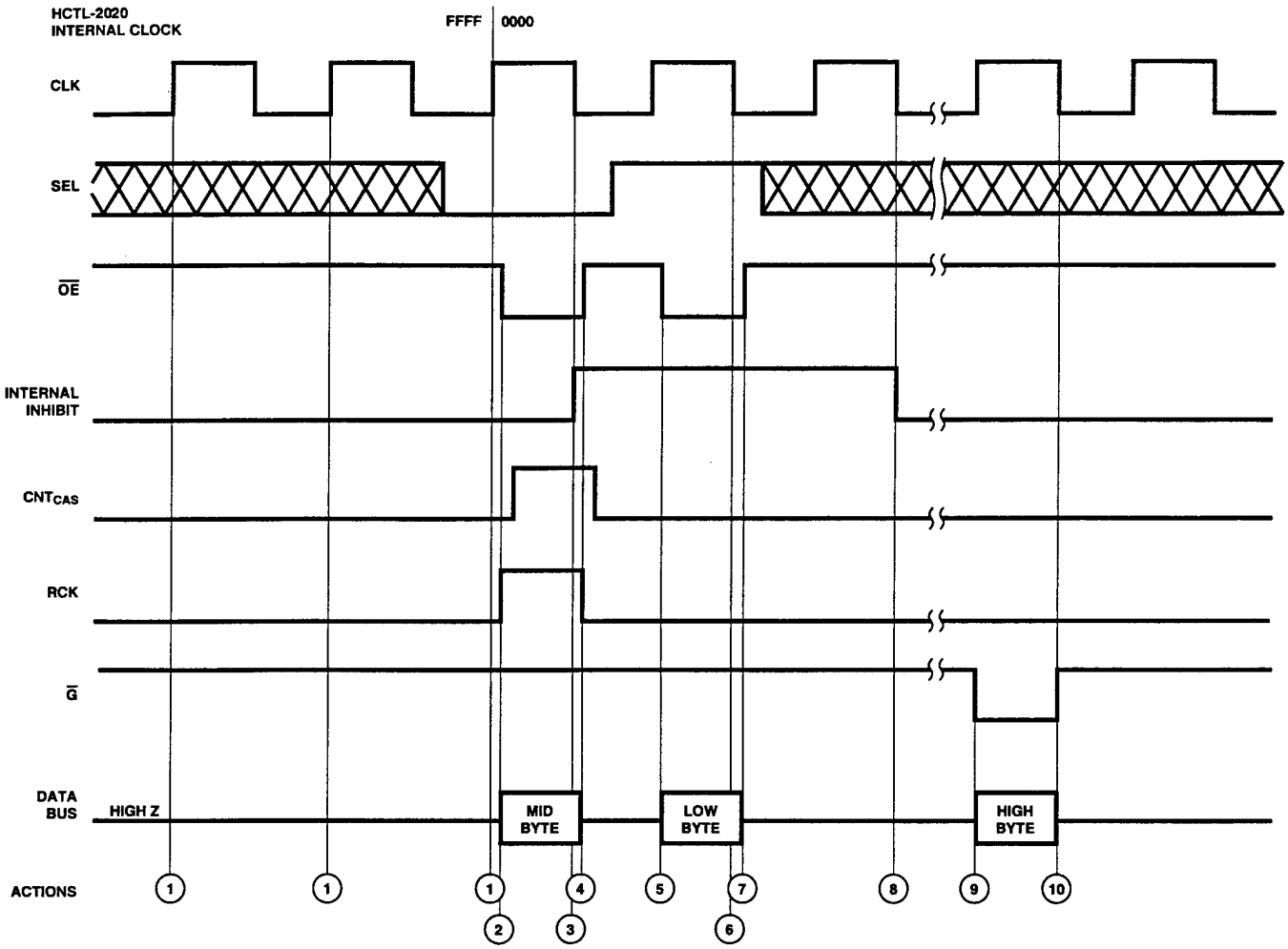


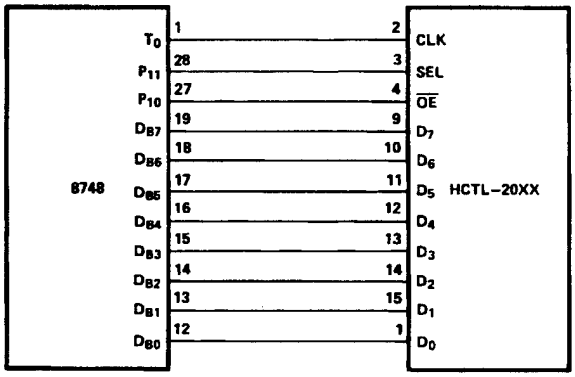
Figure 16. Interface Timing for the 6802/8.

Actions

1. The microprocessor clock output is E. If the internal HCTL-2020 inhibit is not active, new data is transferred from the internal counter to the position data latch.
2. An even address output from the 6802 causes SEL to go low. When E goes high, the address decoder output for the HCTL-2020 $\overline{OE}$ signal goes low. This causes the HCTL-2020 to output the middle byte of the system counter (high byte of the HCTL-2020 counter). This middle byte, FFFFH is available at (2) through (4), the first time OE is low. In this example an overflow has occurred and $\overline{OE}$ has been pulled low to start a read cycle. SEL and OE are gated to give RCK which latches the external high byte, equal to 00H. The falling edge, of the CNT_{CAS} signal counts up the external counter to 0001H.
3. With the first negative edge of the clock after SEL and OE are low the internal latches are inhibited from counting and the 6802 reads the high byte in.
4. $\overline{OE}$ goes high and the data bus goes into a high impedance state.
5. $\overline{OE}$ is low and SEL is high and the low byte is enabled onto the data bus. The low byte is valid through (7).
6. With the first negative edge after $\overline{OE}$ and SEL go high, the first of the two HCTL-2020 inhibit reset conditions is met and the 6802 reads the low byte in.
7. The data bus returns to the high impedance state, when OE goes high.
8. With the first negative edge of the clock after OE goes high, inhibit reset is complete.
9. With the positive going edge of the clock, G is asserted and the external high byte, 00H is available on the data bus from 9 through 10 and the 6802 reads the high byte in at (10).

Interfacing the HCTL-20XX to an Intel 8748

The circuit shown in Figure 17 shows the connections between an HCTL-20XX and an 8748. Data lines D0-D7 are connected to the 8748 bus port. Bits 0 and 1 of port 1 are used to control the $\overline{OE}$ and SEL inputs of the HCTL-20XX respectively. T0 is used to provide a clock signal to the HCTL-20XX. The frequency of T0 is the crystal frequency divided by 3. T0 must be enabled by executing the ENT0 CLK instruction after each system reset, but prior to the first encoder position change. An 8748 program which interfaces to the circuit in Figure 17 is given in Figure 18. The resulting interface timing is shown in Figure 19.



* NOTE: PIN NUMBERS ARE DIFFERENT FOR THE HCTL-2020.

Figure 17. An HCTL-20XX-to-Intel 8748 Interface.

LOC	Object Code	Source Statements	Comments
000	99 00	ANL P1, 00H	Enable output and higher order bits
002	08	INS A, BUS	Load higher order bits into ACC
003	A8	MOVE R0, A	Move data to register 0
004	89 02	ORL P1, 02H	Enable output and lower order bits
006	08	INS A, BUS	Load order bits into AC
008	A9	MOV R1, A	Move data to register 1
009	89 03	ORL P1, 03H	Disable outputs
00B	93	RETR	Return

Figure 18. A Typical Program for Reading HCTL-20XX with an 8748.

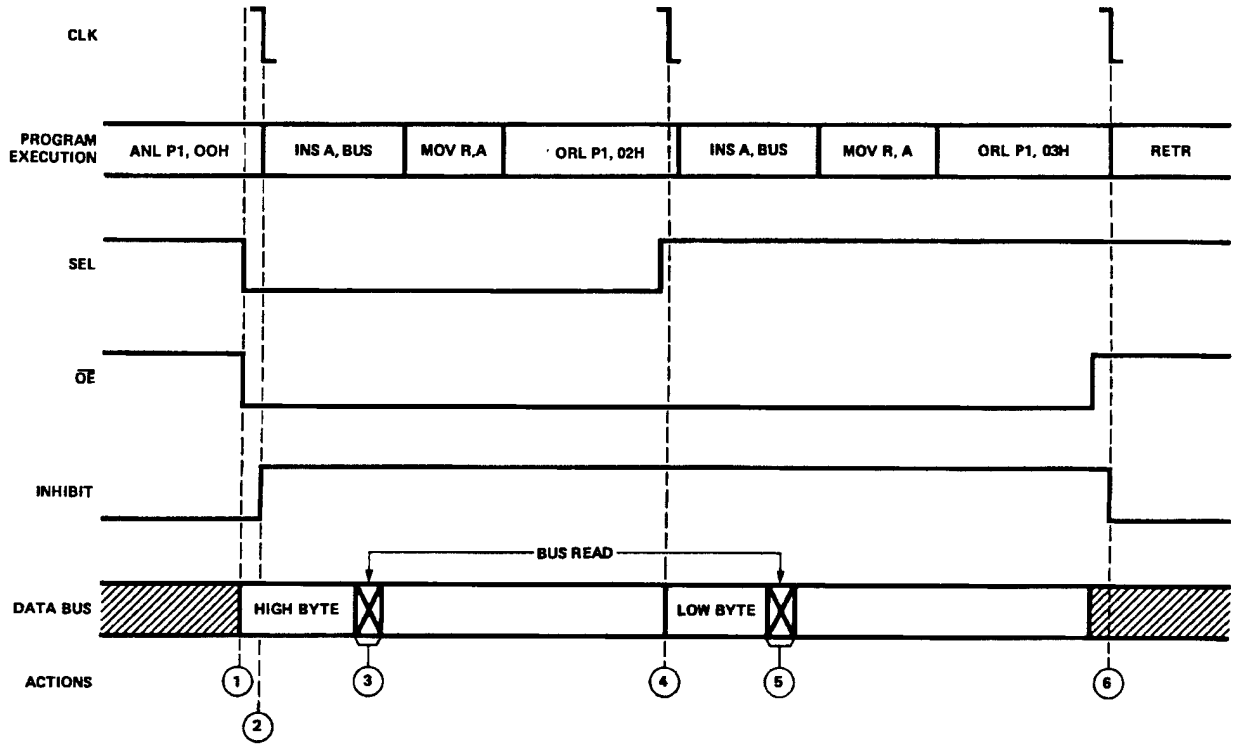


Figure 19. 8748 READ Cycle from Figure 18.

Actions

1. ANL P1, 00H has just been executed. The output of bits 0 and 1 of Port 1 cause SEL and $\overline{OE}$ to be logic low. The data lines output the higher order byte.
2. The HCTL-20XX detects that $\overline{OE}$ and SEL are low on the next falling edge of the CLK and asserts the internal inhibit signal. Data can be read without regard for the phase of the CLK.
3. INS A, BUS has just been executed. Data is read into the 8748.
4. ORL PORT 1, 02H has just been executed. The program sets SEL high and leaves $\overline{OE}$ low by writing the correct values to port 1. The HCTL-20XX detects $\overline{OE}$ is low and SEL is high on the next falling edge of the CLK, and thus the first inhibit reset condition is met.
5. INS A, BUS has just been executed. Lower order data bits are read into the 8748.
6. ORL P1, 03H has just been executed. The HCTL-20XX detects $\overline{OE}$ high on the next falling edge of CLK. The program sets $\overline{OE}$ and SEL high by writing the correct values to port 1. This causes the data lines to be tristated. This satisfies the second inhibit and reset condition. On the next rising CLK edge new data is transferred from the counter to the position data latch.

Additional Information from Agilent Technologies

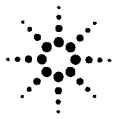
Application briefs are available from the factory. Please contact your local Agilent sales representative for the following.

M027 Interfacing the HCTL-20XX to the 8051

M019 Commonly Asked Questions about the HCTL-2020 and Answers

M020 A Simple Interface for the HCTL-2020 with a 16-bit DAC without Using a Processor

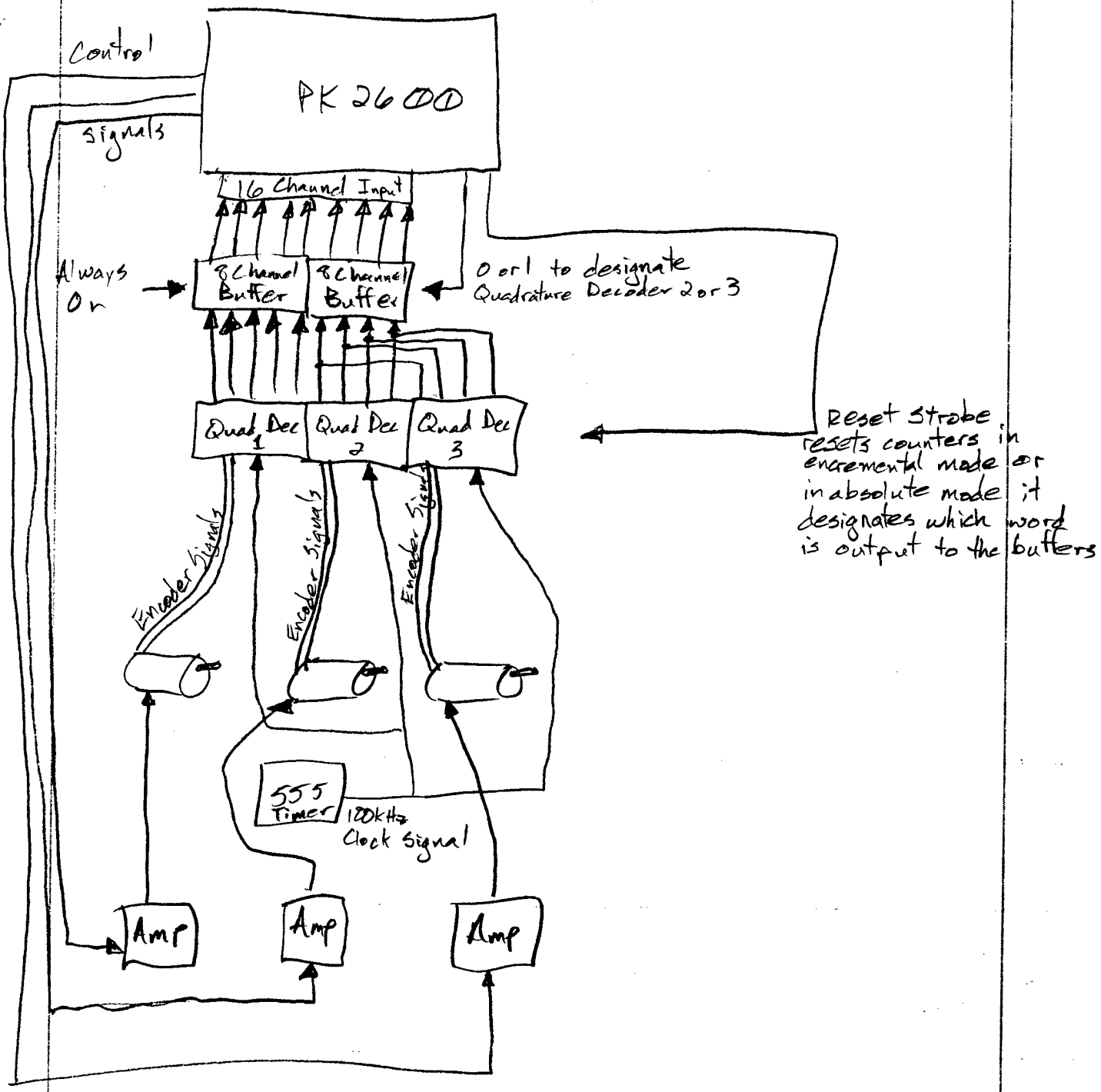
M023 Interfacing the MC68HCII to the HCTL-2020



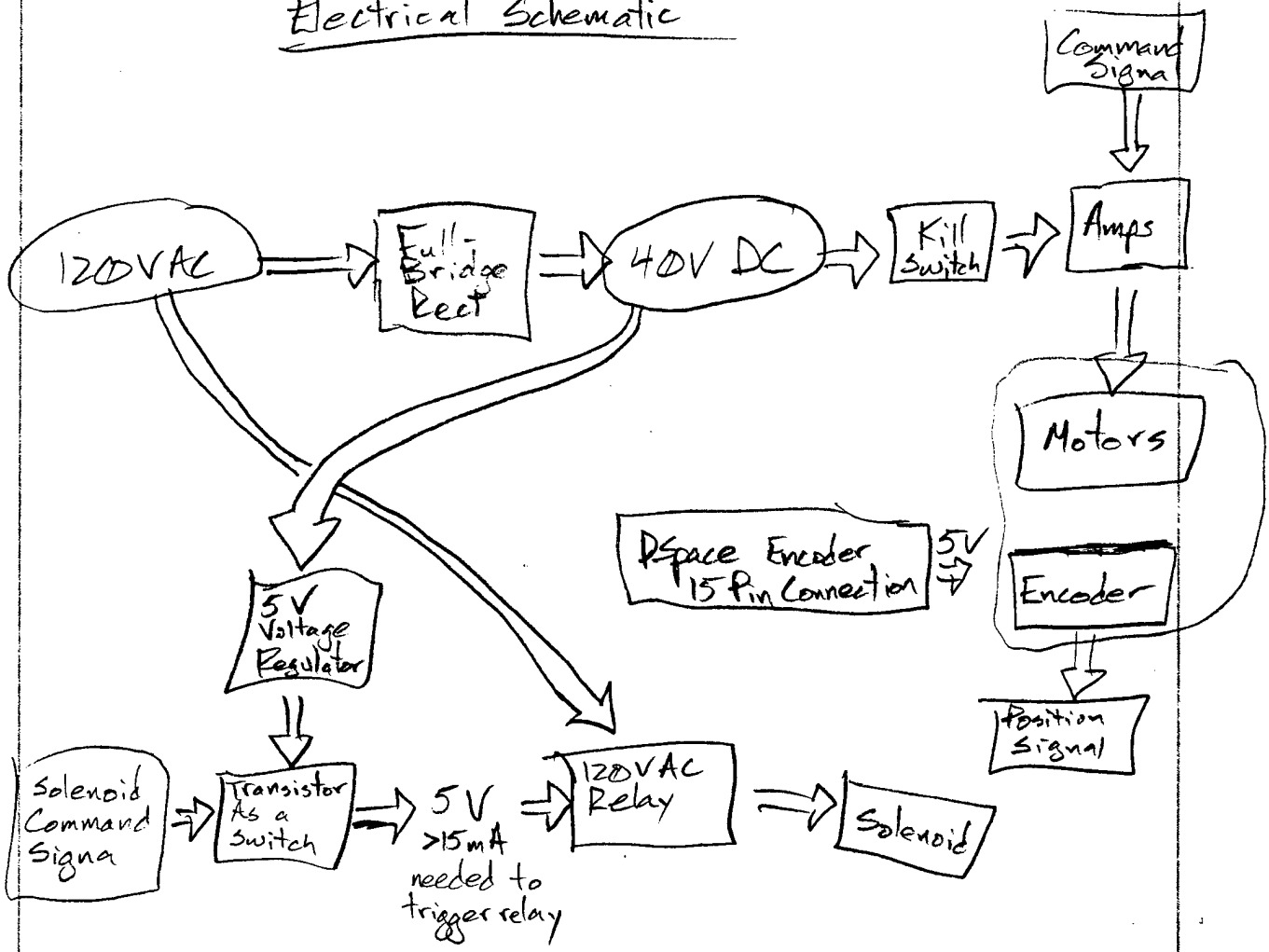
Agilent Technologies
Innovating the HP Way

www.semiconductor.agilent.com

Obsoletes 5091-9974E (3/94)
5965-5894E (11/99)

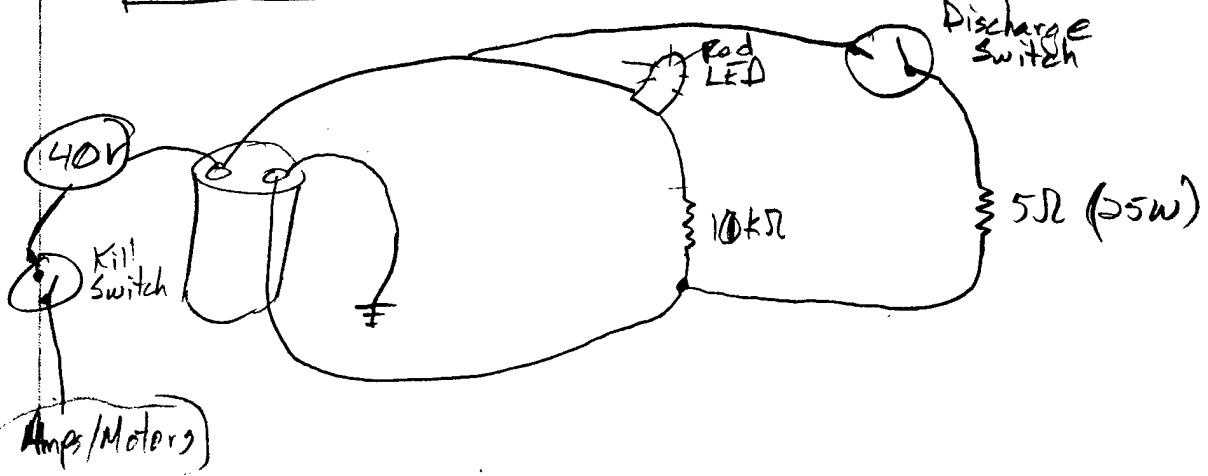

Control
signals
PK 2600
16 Channel Input
Always On
8 Channel Buffer
8 Channel Buffer
0 or 1 to designate Quadrature Decoder 2 or 3
Quad Dec 1
Quad Dec 2
Quad Dec 3
Reset Strobe resets counters in encremental mode or in absolute mode it designates which word is output to the buffers
Encoder Signals
Encoder Signals
Encoder Signals
555 Timer
100kHz Clock Signal
Amp
Amp
Amp

Electrical Schematic



There's a large capacitor in the rectifier, to reduce ripple, when the power is turned off, it holds a charge.

Below is a diagram of the capacitor charge indicator and a safe discharge button - only to be used when power is off.



Bibliography

1. Northrop, M.J., *Parts Feeding with a Throwing Robot.* 1999.
2. Aboaf, E.W., C.G. Atkeson, and D.J. Reinkensmeyer, *Task-Level Robot Learning.* 1988 IEEE International Conference on Robotics and Automation, 1988. **2**: p. 1309-1310.
3. Kato, N., K. Matsuda, and T. Nakamura, *Adaptive Control for a Throwing Motion of a 2 DOF Robot.* 1996 4th International Workshop on Advanced Motion Control, 1996. **1**: p. 203-207.
4. Fu, K.S., R.C. Gonzalez, and C.S.G. Lee, *Robotics Control, Sensing, Vision, and Intelligence.* 1987: p. 98-102.
5. Hollerbach, J.M., *Introduction to Robotics Fall 2000 Course Notes.* 2000: p. Chapter 10 pp 12-17.
6. Craig, J.J., *Introduction to Robotics Mechanics and Cotnrol Second Edition.* 1989: p. 310-321.
7. Wolf, S. and R.F.M. Smith, *Student Reference Manual for Electronic Instrumentation Laboratories.* 1190.